Laporan Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Implementasi Algoritma UCS dan A untuk Menentukan Lintasan Terpendek*

Kelompok 30

Anggota Kelompok:

Ulung Adi Putra	13521122
Mohammad Rifqi Farhansyah	13521166

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

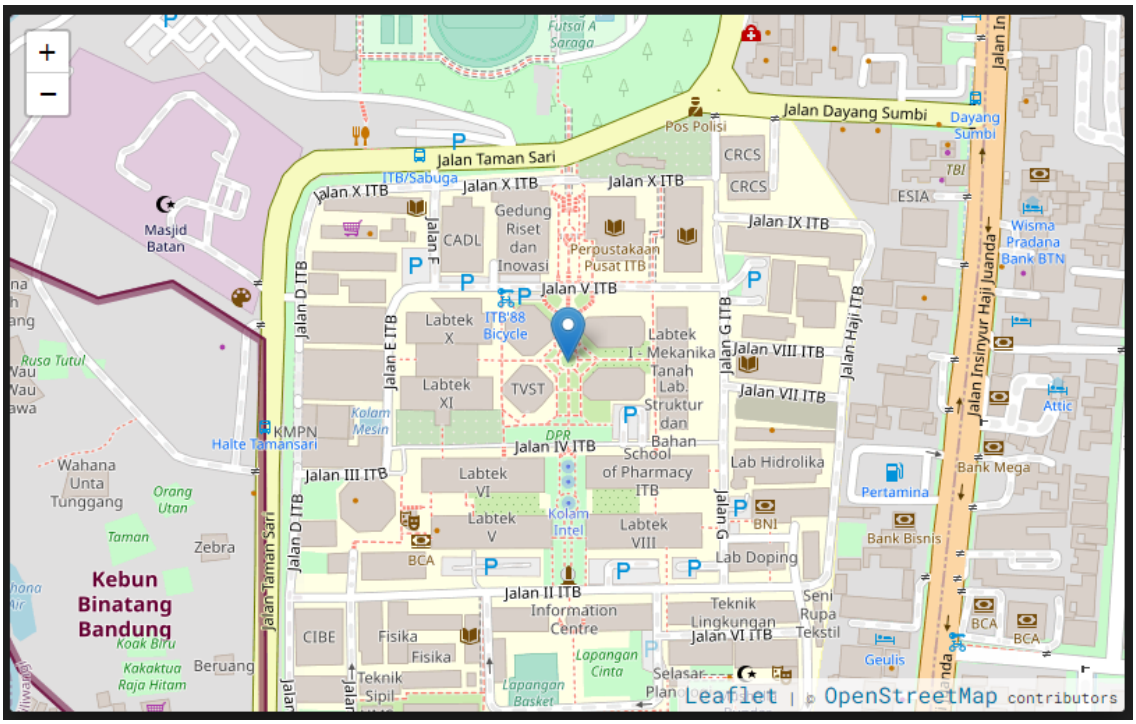
Daftar Isi

Daftar Gambar

BAB I

1.1. Latar Belakang

Kita seringkali dihadapkan pada permasalahan untuk menentukan rute terpendek dari satu titik ke titik lainnya. Permasalahan ini muncul karena keterbatasan-keterbatasan seperti tenaga, waktu, dan lain-lain yang menuntut penentuan strategi terbaik untuk mengoptimalkan sumber daya yang ada. Misalnya, ketika kita harus mengikuti kelas IF2211 – Strategi Algoritma pada hari Rabu, pukul 13.00 WIB, tetapi kita baru bangun tidur jam 12.59. Kita mungkin akan berpikir bahwa mengikuti kelas Stima pada hari itu adalah hal yang mustahil. Namun, dengan strategi pemilihan rute yang tepat, hal itu mungkin saja bisa terwujud.



***Gambar 1.1.1**. Ilustrasi Peta ITB*
Sumber: Dokumen Penulis

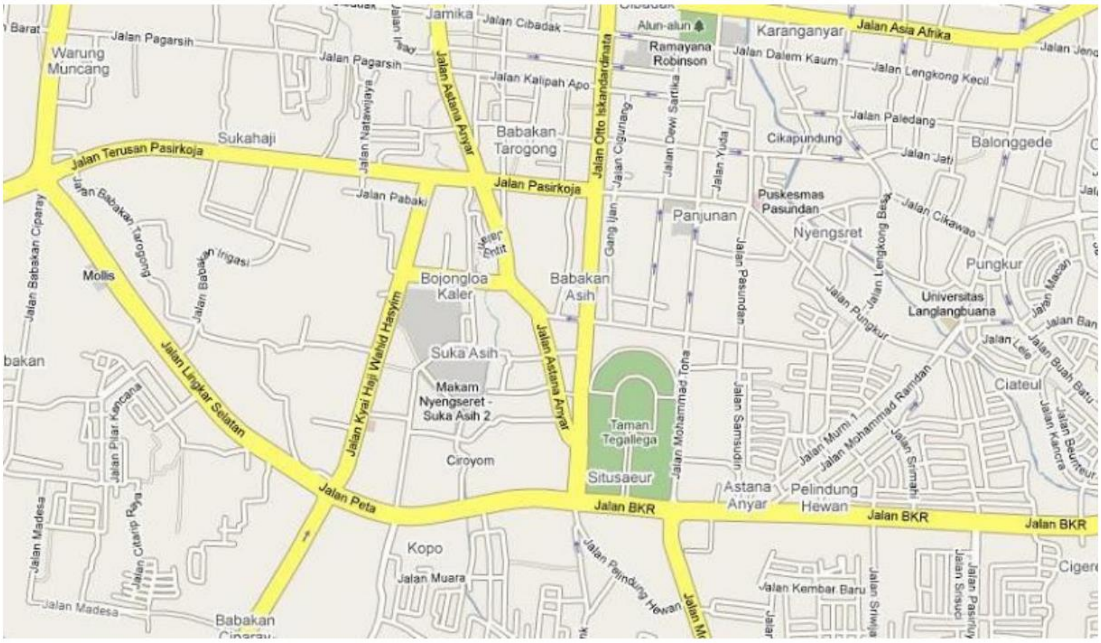
Implementasi algoritma UCS dan A* pada pencarian rute terbaik sangat berguna untuk menyelesaikan permasalahan seperti yang disebutkan di atas. Algoritma UCS (Uniform Cost Search) adalah salah satu algoritma pencarian rute terpendek yang bekerja dengan menghitung biaya untuk mencapai setiap simpul di graf dan memilih simpul dengan biaya terkecil sebagai simpul berikutnya yang akan dikunjungi. Algoritma ini cocok untuk digunakan pada graf yang memiliki bobot yang sama pada setiap simpul.

Sementara itu, algoritma A* (atau A star) merupakan pengembangan dari algoritma UCS dengan tambahan heuristik yang digunakan untuk memperkirakan biaya yang diperlukan untuk mencapai tujuan dari simpul saat ini. Heuristik ini berupa estimasi jarak garis lurus antara simpul saat ini dan tujuan. Algoritma A* memilih simpul dengan biaya terkecil ditambah estimasi heuristik terkecil sebagai simpul berikutnya yang akan dikunjungi.

Kedua algoritma ini sangat berguna untuk menentukan rute terpendek pada graf yang merepresentasikan peta jalan, seperti pada tugas kecil 3 yang diminta untuk menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dengan menggunakan kedua algoritma ini, kita dapat menentukan rute terpendek antara dua titik dengan efisien, mengoptimalkan penggunaan sumber daya yang tersedia, dan meminimalkan waktu dan tenaga yang diperlukan untuk mencapai tujuan.

1.2. Deskripsi Tugas

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.



Gambar 1.2.1. *Ilustrasi Peta Google Map*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil3-Stima-2023.pdf

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.

1.3. Spesifikasi Program

Spesifikasi Tugas Kecil 3 IF2211 - Strategi Algoritma 2023 adalah sebagai berikut:

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.

1.3.1. Spesifikasi Bonus dan Peta

Bonus: Bonus nilai diberikan jika dapat menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta (berupa jalan yang diberi warna). Simpul graf diperoleh dari peta (menggunakan API Google Map) dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Euclidean.

Peta jalan yang digunakan sebagai kasus uji adalah:

1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara.
2. Peta jalan sekitar Alun-alun Bandung.
3. Peta jalan sekitar Buahbatu atau Bandung Selatan.
4. Peta jalan sekitar Buahbatu atau Bandung Selatan.

1.3.2. Spesifikasi Laporan

Berkas yang dikumpulkan adalah laporan format PDF yang berisi:

1. Deskripsi persoalan.
2. Kode program.
3. Peta/graf input, output screenshoot peta yang memperlihatkan lintasan terpendek untuk sepasang simpul, tampilkan hasil untuk beberapa lintasan terpendek di kota Bandung atau kota lainnya yang kalian suka.
4. Alamat github/google drive tempat kode sumber program diletakkan jika perlu dieksekusi oleh asisten.
5. Kesimpulan, komentar, dll.

Spesifikasi lebih detail terkait Tugas Kecil 3 IF2211 – Strategi Algoritma 2023 dapat diakses melalui pranala berikut ini.

BAB II
Landasan Teori

2.1. Graph

Graph adalah suatu struktur data yang terdiri dari kumpulan simpul (node/vertex) yang terhubung oleh kumpulan garis (edge). Secara formal, graph dapat didefinisikan sebagai G = (V, E), di mana V adalah himpunan simpul dan E adalah himpunan edge yang menghubungkan simpul-simpul tersebut. Graph dapat digunakan untuk merepresentasikan berbagai macam hal di dalam dunia nyata, seperti jaringan jalan, jaringan sosial, jaringan listrik, dan sebagainya. Graph juga sering digunakan dalam berbagai macam aplikasi di bidang computer science, seperti algoritma pencarian jalur terpendek, algoritma pengelompokan data, dan sebagainya.

Ada beberapa jenis graph yang umum dikenal, di antaranya adalah:

- Directed graph (digraph): graph yang edge-nya memiliki arah, sehingga mewakili hubungan yang bersifat satu arah.
- Undirected graph: graph yang edge-nya tidak memiliki arah, sehingga mewakili hubungan yang bersifat dua arah.
- Weighted graph: graph yang setiap edge-nya diberi bobot atau nilai tertentu yang mewakili jarak atau biaya untuk mencapai simpul tujuan.
- Connected graph: graph yang semua simpulnya terhubung oleh satu atau lebih edge, sehingga terdapat jalur yang menghubungkan antara setiap simpul.
- Disconnected graph: graph yang tidak semua simpulnya terhubung oleh satu atau lebih edge, sehingga terdapat simpul yang tidak dapat dijangkau dari simpul lain.

Penggunaan graph dalam implementasi algoritma seperti UCS dan A* sangat penting, karena keduanya membutuhkan representasi graf untuk mencari jalur terpendek dari satu simpul ke simpul lainnya. Dengan merepresentasikan jaringan jalan atau jalur lainnya dalam bentuk graph, kita dapat mengaplikasikan algoritma-algoritma tersebut untuk menemukan jalur terpendek dengan mempertimbangkan bobot pada edge-edge yang menghubungkan simpul-simpul tersebut.

2.2. Uniform Cost Search (UCS)

Uninformed search atau blind search adalah jenis algoritma pencarian yang tidak menggunakan informasi tentang sifat struktur masalah yang sedang dicari solusinya. Artinya, algoritma ini tidak menggunakan pengetahuan apapun tentang target pencarian selain informasi yang diberikan oleh definisi masalah itu sendiri. Sebagai contoh, uninformed search hanya mengetahui himpunan keadaan (state space) yang mungkin dilalui dalam mencari solusi.

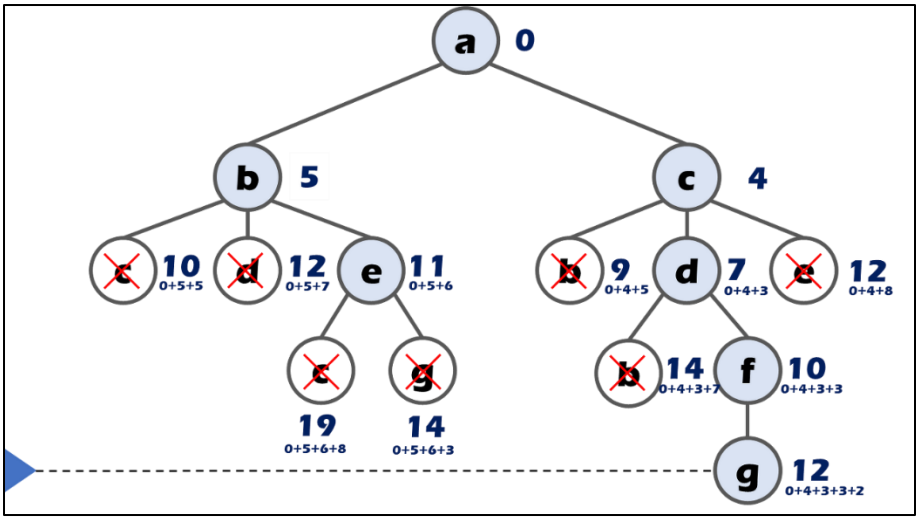
Terdapat beberapa jenis uninformed search, yaitu depth-first search (DFS), breadth-first search (BFS), uniform-cost search (UCS), dan iterative deepening search (IDS). DFS dan BFS adalah dua algoritma yang paling umum digunakan dalam uninformed search. Pada dasarnya, kedua algoritma ini mencari solusi dengan cara melewati semua kemungkinan state space. DFS melakukan pencarian pada satu jalur hingga mencapai simpul terdalam sebelum mencari jalur lain, sedangkan BFS mencari semua simpul pada jarak yang sama dari simpul awal sebelum melanjutkan ke simpul selanjutnya pada jarak yang lebih jauh.

Sementara itu, UCS adalah algoritma uninformed search yang berfokus pada biaya yang dikeluarkan untuk mencapai setiap simpul. Algoritma ini memperhitungkan biaya saat mencari solusi, sehingga meskipun pada akhirnya tidak menemukan solusi terpendek, namun biaya yang dikeluarkan selama pencarian tetap dijaga seminimal mungkin. Hal ini dilakukan dengan memprioritaskan simpul yang memerlukan biaya minimum pada setiap tahap pencarian.

2.3. Uniform Cost Search (UCS)

Uniform Cost Search (UCS) adalah algoritma pencarian graf yang merupakan perluasan dari Breadth First Search (BFS). Seperti pada BFS, UCS juga mencari jalur terpendek dari simpul awal ke simpul tujuan, tetapi UCS tidak hanya mempertimbangkan jarak antara simpul tetangga dengan simpul awal, tetapi juga mempertimbangkan biaya untuk mencapai simpul tersebut. Biaya ini dapat diwakili oleh bobot pada setiap edge dalam graf. Dalam UCS, setiap simpul dalam graf akan dianggap sebagai sebuah state. Algoritma ini mempertahankan sebuah priority queue untuk mengevaluasi simpul yang akan diproses selanjutnya berdasarkan biaya terkecil. Selama proses, jika ditemukan jalur baru yang lebih pendek ke sebuah simpul yang sudah dievaluasi sebelumnya, maka jalur tersebut akan diupdate.

Algoritma UCS memiliki performa yang lebih baik dibandingkan BFS dalam kasus ketika setiap edge memiliki bobot yang sama, atau ketika bobot edge tidak memiliki pola tertentu. Namun, algoritma ini cenderung menghasilkan banyak simpul yang dievaluasi, terutama dalam kasus dengan bobot edge yang bervariasi. Waktu eksekusi UCS tergantung pada banyaknya simpul yang perlu dievaluasi. Dalam kasus terburuk, algoritma ini memerlukan waktu yang eksponensial dengan jumlah simpul. Namun, jika jumlah simpul yang dievaluasi sedikit, algoritma ini dapat memiliki performa yang sangat baik. Dalam implementasi UCS, kita perlu memperhatikan bahwa algoritma ini tidak selalu dapat menemukan jalur terpendek jika terdapat siklus dengan bobot negatif dalam graf. Karena setiap edge dalam graf memiliki bobot positif atau 0, maka algoritma ini aman digunakan dalam kasus-kasus tersebut.



***Gambar 2.3.1**. Ilustrasi Uniform Cost Search (UCS)*
Sumber: https://img1.daumcdn.net/thumb/R1280x0/?scode=mtistory2&fname=https:%2F%2Fblog.kakaocdn.net%2Fdn%2FsNImR%2FbtrpPwmrSdL%2F994fQe7JXEKXo3kt20kdL0%2Fimg.png

2.4. Heuristic Search

Heuristic search atau pencarian heuristik adalah metode pencarian solusi yang memanfaatkan pengetahuan atau informasi tambahan (heuristik) untuk memandu pencarian. Metode ini berbeda dengan uninformed search yang hanya mengandalkan informasi yang diberikan langsung oleh masalah yang dihadapi. Dalam heuristic search, sebuah fungsi heuristik digunakan untuk mengevaluasi setiap simpul pada setiap tahap pencarian dan memberikan perkiraan biaya atau nilai heuristik untuk mencapai solusi akhir. Fungsi heuristik ini didasarkan pada pengetahuan atau informasi tambahan yang tersedia tentang masalah, seperti jarak euclidean antara dua simpul pada graf, estimasi waktu perjalanan, atau estimasi biaya. Heuristic search umumnya lebih efektif daripada uninformed search karena dapat mengurangi jumlah

simpul yang dieksplorasi. Namun, heuristic search juga memiliki kelemahan yaitu tergantung pada kualitas heuristik yang digunakan. Fungsi heuristik yang buruk dapat menyebabkan pencarian terjebak pada simpul-simpul yang tidak relevan dan menghasilkan solusi yang kurang optimal.

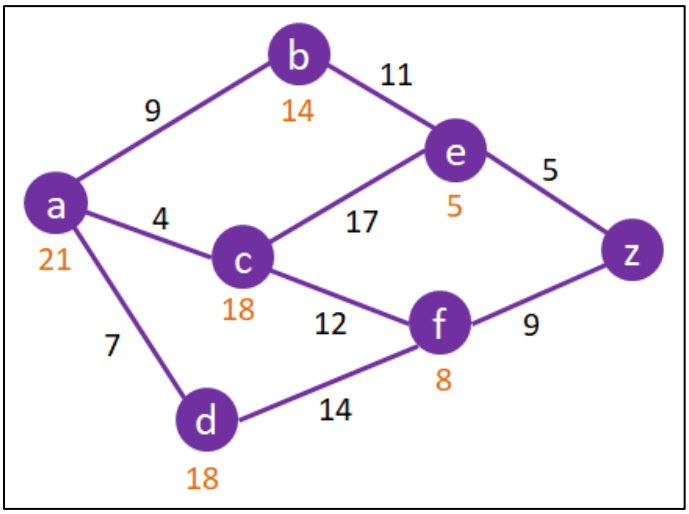
2.5. Algoritma A*

Algoritma A* merupakan algoritma pencarian jalur terpendek yang digunakan pada graf dengan bobot yang berbeda-beda pada setiap edge-nya. Algoritma ini merupakan pengembangan dari algoritma UCS (Uniform Cost Search) dengan tambahan konsep heuristic. Heuristic pada A* digunakan untuk memperkirakan jarak sisa yang harus ditempuh dari suatu simpul ke simpul tujuan, sehingga algoritma A* dapat lebih efektif dalam mencari jalur terpendek.

Cara kerja algoritma A* adalah dengan menggabungkan konsep dari Dijkstra's Algorithm (atau UCS) dan heuristic. Dijkstra's Algorithm digunakan untuk mencari jalur terpendek pada graf dengan bobot yang sama pada setiap edge-nya, sedangkan heuristic digunakan untuk memperkirakan jarak sisa yang harus ditempuh dari suatu simpul ke simpul tujuan.

Pada setiap iterasi, algoritma A* mengambil simpul dengan f(n) terkecil, dimana f(n) = g(n) + h(n), dan g(n) adalah jarak terpendek dari simpul awal ke simpul n, sedangkan h(n) adalah heuristic yang memperkirakan jarak sisa yang harus ditempuh dari simpul n ke simpul tujuan. Kemudian, algoritma A* memeriksa semua tetangga dari simpul tersebut dan menghitung f(n) untuk setiap tetangganya. Jika f(n) lebih kecil dari f(n) sebelumnya, maka simpul tetangga tersebut dimasukkan ke dalam daftar simpul yang akan diproses. Algoritma A* akan berhenti ketika simpul tujuan ditemukan atau ketika tidak ada simpul lagi yang dapat diproses. Jalur terpendek dari simpul awal ke simpul tujuan dapat dihitung dengan mengikuti jalur yang telah ditandai selama proses pencarian.

Algoritma A* termasuk ke dalam kategori algoritma pencarian jalur terpendek dengan heuristic, sehingga keefektifannya sangat tergantung pada kualitas dari heuristic yang digunakan. Heuristic yang buruk dapat mengakibatkan algoritma A* membutuhkan waktu yang lebih lama untuk menemukan jalur terpendek atau bahkan menghasilkan jalur yang suboptimal. Oleh karena itu, pemilihan heuristic yang tepat dan efektif sangat penting dalam penggunaan algoritma A*.



Gambar 2.5.1. *Ilustrasi Algoritma A**
Sumber: https://www.101computing.net/a-star-search-algorithm/

BAB III
Analisis Pemecahan Masalah

3.1 Langkah-langkah Penyelesaian Masalah

3.1.1 Algoritma Uniform Cost Search (UCS)

Algoritma UCS adalah algoritma pencarian jalur terpendek yang bekerja dengan mempertimbangkan biaya yang diperlukan untuk mencapai setiap simpul pada graf. Untuk pemecahan masalah pencarian rute terdekat, langkah-langkah yang dapat dilakukan dengan algoritma UCS adalah sebagai berikut:

1. Ambil simpul awal sebagai simpul yang belum dikunjungi dan tambahkan ke dalam antrian prioritas dengan nilai biaya awal 0.
2. Selama antrian prioritas tidak kosong, lakukan langkah berikut:
    - Keluarkan simpul dengan biaya terendah dari antrian prioritas dan tandai sebagai simpul yang telah dikunjungi.
    - Jika simpul tersebut adalah simpul tujuan, maka kembalikan jalur yang telah ditemukan dari simpul awal ke simpul tujuan.
    - Untuk setiap simpul tetangga yang belum dikunjung, hitung biaya yang diperlukan untuk mencapainya dari simpul awal melalui simpul yang sedang diproses. Kemudian tambahkan simpul tetangga tersebut ke dalam antrian sesuai dengan nilai biaya.
3. Jika antrian prioritas kosong dan simpul tujuan belum ditemukan, maka jalur yang dicari tidak ada.

Pada kode kelompok kami, langkah-langkah pemecahan masalah dengan algoritma UCS telah diterapkan dengan menggunakan antrian prioritas dan kelas parrentNode yang merepresentasikan simpul pada graf. Pertama, simpul awal dan simpul tujuan diambil dari elemen yang dipilih pada tampilan web dan diubah menjadi indeks pada graf dengan menggunakan fungsi getIndex(). Selanjutnya, simpul awal dimasukkan ke dalam antrian prioritas dengan biaya awal 0.

Selama antrian prioritas tidak kosong, simpul dengan biaya terendah diambil dari antrian prioritas dan ditandai sebagai simpul yang telah dikunjungi. Jika simpul tersebut adalah simpul tujuan, maka jalur yang telah ditemukan dari simpul awal ke simpul tujuan dikembalikan. Jika simpul yang sedang diproses bukan simpul tujuan, maka dilakukan pengujian untuk setiap simpul tetangga yang belum dikunjungi. Selanjutnya, simpul tetangga tersebut dimasukkan ke dalam antrian prioritas sesuai dengan nilai biaya.

Jika antrian prioritas kosong dan simpul tujuan belum ditemukan, maka jalur yang dicari tidak ada. Setelah proses selesai, jalur terpendek yang telah ditemukan ditampilkan pada tampilan web dengan menggunakan fungsi drawPath().

3.1.2 Algoritma A*

Algoritma A* adalah algoritma pencarian jalur terpendek yang menggabungkan heuristik (perkiraan jarak terpendek) dengan algoritma Djikstra. Berikut adalah langkah-langkah pemecahan masalah dengan algoritma A* dalam penyelesaian masalah pencarian rute terdekat:

1. Inisialisasi
    - Deklarasikan variabel-variabel yang diperlukan seperti node awal, node tujuan, array node belum dikunjungi (unvisited), nilai fValue (perkiraan jarak terpendek dari node saat ini ke node tujuan), dan nilai cost (jarak antara node saat ini dan node yang terhubung).
    - Tentukan node awal dan node tujuan.
    - Buat objek untuk node awal dengan nilai prev (node sebelumnya), fValue, dan cost yang belum terdefinisi.
    - Masukkan node awal ke dalam array unvisited.
2. Iterasi

- Selama array unvisited masih belum kosong, lakukan hal berikut:
    - Cari node dengan fValue terkecil dari array unvisited.
    - Ambil node tersebut dan hapus dari array unvisited.
    - Hitung nilai gValue baru dengan menambahkan nilai cost dari node sebelumnya ke node yang sedang dipertimbangkan.
    - Jika node yang sedang diperiksa adalah node tujuan, maka kembalikan jalur yang telah ditemukan.
    - Jika tidak, tambahkan node yang terhubung dengan node saat ini ke dalam array unvisited dengan nilai prev, cost, dan fValue yang baru dihitung berdasarkan heuristik dan gValue yang baru.

3. Backtracking
    - Jika node yang sedang diperiksa adalah node tujuan, maka kembalikan jalur yang telah ditemukan dengan melakukan backtracking (menelusuri node sebelumnya dari node tujuan).
    - Jika tidak ditemukan jalur yang valid, kembalikan null atau nilai yang menunjukkan bahwa tidak ada jalur yang ditemukan.
4. Visualisasi
    - Jika jalur ditemukan, gambarkan jalur tersebut pada peta atau tampilan visual lainnya.

Dalam implementasi kode, langkah-langkah tersebut dijalankan dalam dua fungsi, yaitu Astar dan Astaralgorithm. Fungsi Astar memanggil fungsi Astaralgorithm dengan parameter-parameter yang diperlukan, kemudian memutar urutan jalur dan menggambar jalur pada peta. Fungsi Astaralgorithm melakukan langkah-langkah iterasi dan backtracking untuk mencari jalur terpendek.

BAB IV
Implementasi dan Pengujian

4.1 Implementasi Program

4.1.1. Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>UCS and A* Map</title>
    <link rel="icon" href="map.png" type="image/x-icon">
    <link rel="stylesheet" href="style.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto+Mono">
    <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
    integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZC
hSR7A=="
    crossorigin=""/>
    <!-- Make sure you put this AFTER Leaflet's CSS -->
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"
    integrity="sha512-
XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3
ynxwA=="
    crossorigin=""></script>
</head>
<body>
    <h1><span class="logo"></span>UCS and A* Map</h1>
    <div>
        <input type="file" accept=".json" onchange="readDataset(event)">
        <label for="view">Set View:</label>
        <select name="view" id="view" onchange="setView()">
            <option value="">-- Choose View --</option>
            <option value="itb">ITB</option>
            <option value="dago">Dago</option>
            <option value="magelang">Magelang</option>
            <option value="buahbatu">Buah Batu</option>
            <option value="Purworejo">Purworejo</option>
        </select>
    </div>
    <div id="mapid"></div>
    <div>
        <p id="sum-path"></p>
    </div>
    <div>
        <label for="from-node">Start Node:</label>
        <select name="from-node" id="from-node" onchange="runAlgorithm()">
        </select>
        <label for="to-node">Goal Node:</label>
        <select name="to-node" id="to-node" onchange="runAlgorithm()">
        </select>
        <label for="algorithm">Algorithm:</label>
        <select name="algorithm" id="algorithm" onchange="runAlgorithm()">
            <option value="astar">A* Algorithm</option>
            <option value="ucs">Uniform Cost Search</option>
        </select>
```

```
    </div>
    <script type="text/javascript" src="leaflet.textpath.js"></script>
    <script type="text/javascript" src="index.js"></script>
</body>
</html>
```

4.1.2. Style.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Roboto Mono', monospace;
    font-size: 16px;
}

body {
    background-color: #222;
    color: #fff;
    display: flex;
    flex-direction: column;
    align-items: center;
}

h1 {
    font-size: 48px;
    margin-top: 32px;
    margin-bottom: 64px;
}

input[type="file"] {
    background-color: #444;
    border: none;
    border-radius: 4px;
    color: #fff;
    padding: 12px;
    margin-bottom: 32px;
    transition: background-color 0.3s ease-in-out;
}

input[type="file"]:hover {
    cursor: pointer;
    background-color: #555;
}

#mapid {
    height: 500px;
    width: 800px;
    margin-bottom: 32px;
    border-radius: 4px;
    overflow: hidden;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.4);
    opacity: 0;
    transform: perspective(800px) rotateX(10deg) scale(0.9);
    transition: opacity 0.3s ease-in-out, transform 0.3s ease-in-out;
}

#mapid.show {
    opacity: 1;
    transform: perspective(800px) rotateX(0deg) scale(1);
}
```

```
#sum-path {
    font-size: 24px;
    margin-bottom: 16px;
}

div > p {
    font-size: 24px;
    margin-bottom: 8px;
}

select {
    background-color: #444;
    border: none;
    border-radius: 4px;
    color: #fff;
    padding: 12px;
    margin-bottom: 32px;
}

select option:hover {
    background-color: #555;
}

.loader {
    border: 4px solid #f3f3f3;
    border-top: 4px solid #3498db;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

4.1.3. Index.js

```
//DEFINISI KELAS
class PriorityQueue {
    // Membuat constructor tanpa parameter
    constructor() {
        this.queue = [];
    }
    // Method enqueue digunakan untuk menambahkan elemen ke dalam antrian
prioritas.
    enqueue(node) {
        this.queue.push(node);
        this.queue.sort((a, b) => a.cost - b.cost); // Mengurutkan elemen
berdasarkan cost dari terkecil ke terbesar.
    }
    // Method dequeue digunakan untuk mengambil elemen terdepan dari antrian
prioritas.
    dequeue() {
        return this.queue.shift();
    }
    // Method isEmpty digunakan untuk mengecek apakah antrian prioritas kosong
atau tidak.
```

```
    isEmpty() {
        return this.queue.length === 0;
    }
}
class parrentNode {
    // Membuat constructor dengan parameter id, parent, dan cost
    constructor(id, parrent, cost) {
        this.id = id; // variabel untuk menyimpan id node
        this.parrent = parrent; // variabel untuk menyimpan node parrent
        this.cost = cost; // variabel untuk menyimpan cost dari node saat ini
    }
}
class Node {
    // Membuat constructor dengan parameter name, lat, dan lon
    constructor(name, lat, lon) {
        // Menyimpan nilai parameter name ke dalam properti name
        this.name = name;
        // Menyimpan nilai parameter lat ke dalam properti lat
        this.lat = lat;
        // Menyimpan nilai parameter lon ke dalam properti lon
        this.lon = lon;
    }
    // Membuat method getLatitudeLongitude
    getLatitudeLongitude() {
        // Deklarasi array coordinates yang berisi nilai properti lat dan lon
        const coordinates = [this.lat, this.lon];
        // Mengembalikan array coordinates
        return coordinates;
    }
    // Membuat method findDistance dengan parameter node
    findDistance(node) {
        // Deklarasi variabel distance dan menghitung jarak antara 2 node dengan
menggunakan function calculateDistance dan parameter properti lat dan lon
        const distance = calculateDistance(this.lat, this.lon, node.lat,
node.lon);
        // Mengembalikan nilai distance
        return distance;
    }
}

class Graph {
    constructor() {
        this.nodes = [];
        this.adjacentMatrix = [];
        this.nodeArea = L.layerGroup([]);
        this.edgePaths = L.layerGroup([]);
        this.shortestPath = L.layerGroup([]);
    }
    // fungsi untuk mendapatkan node berdasarkan nama
    getNode(name) {
        return this.nodes.find(node => node.name === name);
    }
    // fungsi untuk menggambar marker node
    drawNodeMarker() {
        for (const node of this.nodes) {
```

```
            const circle = L.circle(node.getLatitudeLongitude(), {radius: 20});
            circle.bindPopup(node.name);
            this.nodeArea.addLayer(circle);
        }
    }
    // fungsi untuk menggambar jalur edge
    drawEdgePath() {
        for (let i = 0; i < this.nodes.length; i++) {
            for (let j = 0; j <= i; j++) {
                if (this.adjacentMatrix[i][j] > 0) {
                    const line =
L.polyline([this.nodes[i].getLatitudeLongitude(),
this.nodes[j].getLatitudeLongitude()]);
                    line.bindPopup(String(this.adjacentMatrix[i][j]));
                    line.setText(String(this.adjacentMatrix[i][j]), {center:
true});
                    this.edgePaths.addLayer(line);
                }
            }
        }
    }
    // fungsi untuk menggambar jalur terpendek
    drawPath(path, map) {
        this.shortestPath.clearLayers();
        let sum = 0;
        for (let i = 0; i < path.length - 1; i++) {
            const from = this.getNode(path[i]);
            const to = this.getNode(path[i + 1]);
            const line = L.polyline([from.getLatitudeLongitude(),
to.getLatitudeLongitude()], {color: 'red'});
            this.shortestPath.addLayer(line);
            this.shortestPath.addTo(map);

            sum += calculateDistance(from.lat, from.lon, to.lat, to.lon);
        }
        const textsum = document.getElementById("sum-path");
        textsum.innerHTML = "Shortest path's distance: " + sum.toString() + "
km";
    }
    // fungsi untuk menggambar node dan edge pada peta
    draw(map) {
        this.drawNodeMarker();
        this.drawEdgePath();
        this.edgePaths.addTo(map);
        this.nodeArea.addTo(map);
    }
    // fungsi untuk membersihkan layer pada peta
    clear() {
        this.shortestPath.clearLayers();
        this.edgePaths.clearLayers();
        this.nodeArea.clearLayers();
    }
    // fungsi untuk mendapatkan indeks node berdasarkan nama
    getIndex(name) {
        return this.nodes.findIndex(x => x.name === name);
```

```
    }
}



//DEFINISI FUNGSI HELPER
// Membuat fungsi calculateDistance dengan parameter latitude1, longitude1,
latitude2, dan longitude2
function calculateDistance(latitude1, longitude1, latitude2, longitude2) {
    // Menetapkan konstanta R dengan nilai 6371 yang merupakan radius bumi dalam
km
    const R = 6371;
    // Mengubah perbedaan latitude menjadi radian dan menyimpannya dalam variabel
_latitude
    const _latitude = derajatToRadian(latitude2 - latitude1);
    // Mengubah perbedaan longitude menjadi radian dan menyimpannya dalam
variabel _longitude
    const _longitude = derajatToRadian(longitude2 - longitude1);
    // Menghitung nilai a menggunakan rumus haversine formula
    const a = Math.sin(_latitude / 2) * Math.sin(_latitude / 2) +
              Math.cos(derajatToRadian(latitude1)) *
Math.cos(derajatToRadian(latitude2)) *
              Math.sin(_longitude / 2) * Math.sin(_longitude / 2);
    // Menghitung nilai c menggunakan rumus haversine formula
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    // Menghitung nilai d (jarak) menggunakan rumus haversine formula dan radius
bumi R
    const d = R * c;
    // Mengembalikan nilai d (jarak)
    return d;
}

// Membuat fungsi derajatToRadian dengan parameter degree
function derajatToRadian(degree) {
    // Menghitung nilai radian dengan rumus degree * Math.PI / 180
    const radian = degree * Math.PI / 180;
    // Mengembalikan nilai radian
    return radian;
}

// fungsi yang akan dijalankan ketika pengguna mengklik peta
function onMapClick(e) {
    alert("You clicked the map at " + e.latlng);
}

//VARIABEL GLOBAL
// Inisialisasi peta dengan koordinat tertentu
let map = L.map('mapid').setView([-6.889295, 107.610365], 17);
function setView() {
    var view = document.getElementById("view").value;
    switch (view) {
        case "itb":
            map.setView([-6.889295, 107.610365], 17);
            break;
        case "dago":
```

```
            map.setView([-6.885172, 107.613724], 17);
            break;
        case "buahbatu":
            map.setView([-6.954339, 107.638979], 17);
            break
        case "magelang":
            map.setView([-7.584627, 110.287199], 17);
            break;
        case "Purworejo":
            map.setView([-7.713049, 110.007964], 17);

        default:
            break;
    }
}
// Membuat objek Graph baru
const graph = new Graph();
// Menambahkan tile layer pada peta
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '© <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(map);
// Menambahkan marker pada peta
L.marker([-6.889295, 107.610365]).addTo(map)
    .bindPopup('ITB')
    .openPopup();
L.marker([-6.885172, 107.613724]).addTo(map)
    .bindPopup('Simpang Dago');
L.marker([-6.954339, 107.638979]).addTo(map)
    .bindPopup('Buah Batu');
L.marker([-7.584627, 110.287199]).addTo(map)
    .bindPopup('Muntilan');

document.getElementById("mapid").classList.add("show");

// Fungsi untuk membaca dataset
const readDataset = (event) => {
    // Buat objek FileReader
    let dataset = new FileReader();
    // Ketika FileReader selesai membaca dataset
    dataset.onload = function(event){
        // Parsing data dari format JSON ke objek JavaScript
        var data = JSON.parse(event.target.result);
        // Panggil fungsi main dengan parameter nodes dan edges dari data
        main(data.nodes, data.edges);
    };
    // Baca file dataset yang diupload oleh pengguna
    dataset.readAsText(event.target.files[0]);
}

// Fungsi untuk menampilkan jalur terpendek antara dua node.
const main = (nodes, edges) => {
    // Menghapus semua node dan edge yang ada pada peta.
    graph.clear();
    // Menghapus elemen pada dropdown list dari node asal dan tujuan.
```

```
    let from = document.getElementById("from-node");
    let goal = document.getElementById("to-node");
    let i=0;
    if (from.length != 0 || goal.length!=0){
        while(from.length!=0 && goal.length!=0){
            from.remove(from.i);
            goal.remove(goal.i);
            i++;
        }
    }
    // Inisialisasi array untuk menyimpan node-node dari graf.
    let nodeArr = [];
    for (let node of nodes) {
        let tnode = new Node(node.name, node.lat, node.lon);
        nodeArr.push(tnode);
    }
    // Tambahkan elemen pada dropdown list untuk node asal dan tujuan.
    for (let node of nodes){
        let elmtfrom = document.createElement("option");
        let elmtto = document.createElement("option");
        elmtfrom.text = node.name;
        elmtto.text = node.name;
        document.getElementById("from-node").add(elmtfrom);
        document.getElementById("to-node").add(elmtto);
    }
    // Set node-node pada graf.
    graph.nodes = nodeArr;
    // Inisialisasi matriks adjacency untuk graf.
    let matrix = edges;
    for(let i = 0; i < graph.nodes.length; i++){
        for(let j = 0; j < i+1; j++){
            if(matrix[i][j] == 1){
                let dist = calculateDistance(nodeArr[i].lat, nodeArr[i].lon,
nodeArr[j].lat, nodeArr[j].lon);
                matrix[i][j] = dist.toPrecision(4)
                matrix[j][i] = dist.toPrecision(4)
            }
        }
    }
    // for(let i=0; i<nodeArr.length; i++){
    //     matrix[i] = [];
    //     for(let j=0; j<nodeArr.length; j++){
    //         matrix[i][j] = 0;
    //     }
    // }
    // Mengisi nilai pada matriks adjacency dengan jarak antar node.
    // for (let edge of edges){
    //     var idx = 0;
    //     let idx_from_found = false;
    //     let idx_to_found = false;
    //     var idx_from=-1;
    //     var idx_to=-1;
    //     while (idx < nodeArr.length && (!idx_from_found || !idx_to_found)){
    //         // Cari index dari node asal pada array nodeArr.
    //         if (edge.from == nodeArr[idx].name && !idx_from_found){
```

```
    //              idx_from = idx;
    //              idx_from_found = true;
    //          }
    //          // Cari index dari node tujuan pada array nodeArr.
    //          if (edge.to == nodeArr[idx].name && !idx_to_found){
    //              idx_to = idx;
    //              idx_to_found = true;
    //          }
    //          idx++;
    //      }
    //      let dist = calculateDistance(nodeArr[idx_from].lat,
nodeArr[idx_from].lon, nodeArr[idx_to].lat, nodeArr[idx_to].lon);
    //      matrix[idx_from][idx_to] = dist.toPrecision(4);
    //      matrix[idx_to][idx_from] = dist.toPrecision(4);
    // }
    // Set matriks adjacency pada graf dan tampilkan pada peta.
    graph.adjacentMatrix = matrix;
    graph.draw(map);
}

// FUNGSI UCS
function ucs(){
    let fromnode = document.getElementById("from-node");
    let goalnode = document.getElementById("to-node");
    let startname = fromnode.options[fromnode.selectedIndex].text;
    let goalname = goalnode.options[goalnode.selectedIndex].text;
    let idx_from = graph.getIndex(startname);
    let idx_to = graph.getIndex(goalname);

    let visited = []
    let queue = new PriorityQueue();
    queue.enqueue(new parrentNode(idx_from, null, 0), 0);
    while(!queue.isEmpty()){
        let node = queue.dequeue();
        let cost = node.cost
        if(node.id === idx_to){
            let path = []
            let currNode = node
            while(currNode !== null){
                console.log(graph.nodes[currNode.id].name)
                path.push(graph.nodes[currNode.id].name)
                currNode = currNode.parrent
            }
            path.reverse();
            graph.drawPath(path, map);
            return path;
        }
        visited[node.id] = true
        for(let i = 0; i < graph.adjacentMatrix[node.id].length; i++){
            if(!visited[i] && graph.adjacentMatrix[node.id][i] !== 0){
                queue.enqueue(new parrentNode(i, node,
Math.abs(graph.adjacentMatrix[node.id][i])+cost),
Math.abs(graph.adjacentMatrix[node.id][i])+cost)
            }
        }
```

```
    }
    return null
}

// FUNGSI A*
function Astar() {
    // Deklarasi variabel
    let fromnode = document.getElementById("from-node");
    let goalnode = document.getElementById("to-node");
    let startname = fromnode.options[fromnode.selectedIndex].text;
    let goal = goalnode.options[goalnode.selectedIndex].text;
    let unvisited = [];
    let start = {
        name: startname,
        prev: null,
        fValue: undefined,
        cost: undefined
    };
    // Memanggil fungsi Astaralgorithm untuk mencari jalur terpendek
    let path = Astaralgorithm(start, goal, 0, unvisited);
    // Memutar urutan jalur
    path.reverse();
    // Menggambar jalur pada peta
    graph.drawPath(path, map);
    // Mengembalikan jalur terpendek
    return path;
}

function Astaralgorithm(current, goal, gValue, unvisited) {
    // Jika node yang sedang diperiksa adalah node tujuan, maka kembalikan jalur
yang telah ditemukan
    if (current.name == goal) {
        let path = [];
        while (current != null) {
            path.push(current.name);
            current = current.prev;
        }
        return path;
    }
    // Mencari indeks node saat ini di dalam graph
    let currIdx = graph.getIndex(current.name);
    // Memasukkan node yang terhubung dengan node saat ini ke dalam array
unvisited
    for (let i = 0; i < graph.nodes.length; i++) {
        if (graph.adjacentMatrix[currIdx][i] > 0) {
            let toVisit = {};
            toVisit.name = graph.nodes[i].name;
            toVisit.prev = current;
            toVisit.cost = Number(graph.adjacentMatrix[currIdx][i]);

            // Menghitung fValue, yaitu perkiraan jarak terpendek dari node saat
ini ke node tujuan melalui node yang sedang dipertimbangkan
            toVisit.fValue =
Number(graph.nodes[i].findDistance(graph.getNode(goal))) + Number(gValue) +
Number(graph.adjacentMatrix[currIdx][i]);
```

```
            unvisited.push(toVisit);
        }
    }
    // Mencari node selanjutnya dengan fValue terkecil
    let minIdx = 0;
    for (let i = 1; i < unvisited.length; i++) {
        if (unvisited[i].fValue < unvisited[minIdx].fValue) {
            minIdx = i;
        }
    }
    // Mengambil node selanjutnya dan menghitung nilai gValue yang baru
    let next = unvisited.splice(minIdx, 1);
    let newG = gValue+next[0].cost;
    // Melakukan rekursi untuk mencari jalur terpendek dari node selanjutnya ke
node tujuan
    return Astaralgorithm(next[0], goal, newG, unvisited);
}

// Fungsi untuk menjalankan algoritma yang dipilih
function runAlgorithm() {
    // Mendapatkan nilai dropdown "algorithm"
    const algorithm = document.getElementById("algorithm").value;
    // Jika nilai dropdown "algorithm" adalah "astar", jalankan algoritma A*
    if (algorithm === "astar") {
        Astar();
    }
    // Jika nilai dropdown "algorithm" adalah "ucs", jalankan algoritma UCS
    else if (algorithm === "ucs") {
        ucs();
    }
}
```

4.2 Penjelasan Struktur Data

Berikut merupakan struktur data beserta spesifikasi program yang digunakan dalam membuat program:

1. Graph

   Struktur data Graph pada program implementasi fungsi A* dan UCS dalam pencarian rute terdekat ini terdiri dari beberapa atribut dan method yang menjelaskan karakteristik dan fungsi dari graph tersebut. Berikut adalah penjelasan atribut dan method dalam struktur data Graph tersebut:

   a) Atribut:

   - nodes : array yang menyimpan objek Node yang merepresantikan setiap node pada graph.
   - adjacentMatrix : array dua dimensi yang merepresentasikan matriks adjacency dari graph. Nilai adjacency matrix adalah jarak (dalam satuan km) antara dua node yang bersebelahan. Jika nilai adjacency matrix bernilai 0, maka tidak ada jalur yang menghubungkan kedua node tersebut.
   - nodeArea : LayerGroup dari Leaflet yang menyimpan semua marker node.
   - edgePaths: LayerGroup dari Leaflet yang menyimpan semua polyline jalur edge antar node.

- shortestPath : LayerGroup dari Leaflet yang menyimpan polyline jalur terpendek yang dihasilkan dari algoritma pencarian rute terdekat.

b) Method:

- Constructor : method yang digunakan untuk inisialisasi atribut pada object Graph.
- getNode(name) : method yang digunakan untuk mendapatkan object Node dari array nodes berdasarkan nama node.
- drawNodeMarker() : method yang digunakan untuk menggambar marker node pada peta Leaflet dengan menggunakan Leaflet.circle. Setiap circle merepresentasikan sebuah node pada graph dan memiliki radius 20 piksel serta label yang berisi nama node.
- drawEdgePath() : method yang digunakan untuk menggambar polyline jalur edge antar node pada peta Leaflet dengan menggunakan Leaflet.polyline. Setiap polyline menunjukkan jalur yang menghubungkan dua node yang bersebelahan pada graph dan memiliki label yang menunjukkan nilai adjacency matrix.
- drawPath(path, map) : method yang digunakan untuk menggambar polyline jalur terpendek pada peta Leaflet dengan menggunakan Leaflet.polyline. Setiap polyline menunjukkan jalur yang merupakan bagian dari jalur terpendek yang dihasilkan darialgoritma pencarian rute terdekat dan memiliki warna merah. Selain itu, method ini juga menghitung jarak total dari jalur terpendek dan menampilkan hasilnya pada sebuah elemen HTML.
- draw(map) : method yang digunakan untuk menggambar semua marker node dan polyline jalur edge pada peta Leaflet. Method ini memanggil method drawNodeMarker() dan drawEdgePath(), serta menambahkan LayerGroup nodeAre dan edgePaths ke dalam peta Leaflet.
- clear() : method yang digunakan untuk membersihkan semua LayerGroup pada peta Leaflet.
- getIndex(name) : method yang digunakan untuk mendapatkan indeks node berdasarkan nama node pada array nodes.

2. Node

   Struktur data Node adalah sebuah class yang merepresentasikan sebuah node dalam sebuah graph. Setiap instance dari class Node memiliki tiga properti yaitu, name, lat, dan lon. Selain itu, Node memiliki beberapa method khusus, antara lain:

   - Constructor() : Method yang digunakan untuk membuat objek Node abru dengan parameter nama, latitude, dan longitude yang akan disimpan pada atribut name, lat, dan lon pada objek Node.
   - getLatitudeLongitude() : Method yang digunakan untuk mengambil nilai latitude dan longitude dari objek Node serta mengembalikan nilai tersebut dalam bentuk array koordinat [lat, lon].
   - findDistance(node) : Method yang digunakan untuk menghitung jarak antara objek Node dengan objek Node lainnya dengan menggunakan formula haversine dan nilai latitude serta longitude yang disimpan pada kedua objek Node. Method ini akan mengembalikan nilai jarak antara kedua objek Node.

3. parrentNode

Struktur data parrentNode adalah sebuah class yang merepresentasikan suatu index node dengan node yang menghasilkan node dengan index tersebut (parrent). Setiap instance dari class parrentNode memiliki tiga properti yaitu, id, parrent, dan cost. Id pada class ini menyatakan suatu index dimana node disimpan dalam graph.nodes, parrent menyatakan suatu parrentNode yang menggenerate node dengan index id, dan cost merupakan representasi dari jarak yang sudah ditempuh untuk mencapai node dengan index id dari titik awal.

4. PriorityQueue

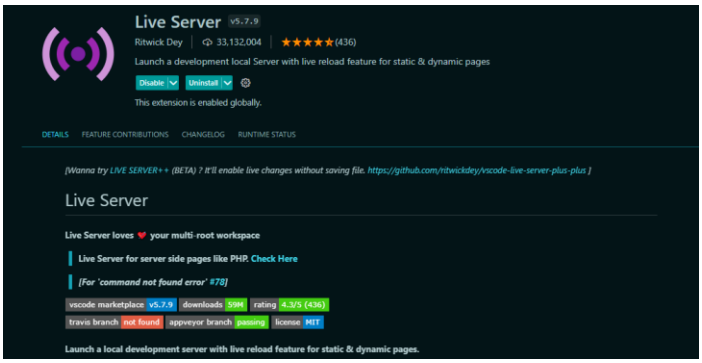
   Struktur data PriorityQueue adalah struktur data yang menyimpan elemen dengan prioritas tertentu dan dapat mengeluarkan elemen dengan prioritas tertinggi terlebih dahulu. Pada implementasi fungsi UCS dalam pencarian rute terdekat, PriorityQueue digunakan untuk menyimpan simpul-simpul yang belum dikunjungi dan diurutkan berdasarkan nilai jarak terpendek dari simpul awal. Berikut adalah penjelasan untuk tiap method dan atribut pada struktur data PriorityQueue:

   - constructor(): method ini digunakan untuk membuat PriorityQueue dengan inisialisasi antrian sebagai array kosong.
   - enqueue(node): method ini digunakan untuk menambahkan simpul yang bertipe parrentNode ke dalam antrian dengan prioritas tertentu. Method ini menambahkan objek yang berisi node ke dalam antrian menggunakan method push(). Kemudian, method sort() dipanggil untuk mengurutkan antrian berdasarkan prioritas.
   - dequeue(): method ini digunakan untuk menghapus dan mengembalikan objek dengan prioritas tertinggi (memiliki cost terendah) dari antrian. Method ini mengembalikan null jika antrian kosong. Jika tidak kosong, method ini menghapus dan mengembalikan objek pertama dalam antrian menggunakan method shift().
   - sort(): method ini digunakan untuk mengurutkan antrian berdasarkan cost. Method ini menggunakan method sort() JavaScript dengan perbandingan cost. Method ini membandingkan setiap elemen pada antrian dan mengurutkannya berdasarkan nilai cost dari parrentNode, dimana nilai cost yang lebih kecil akan diletakan pada urutan yang lebih awal. Method ini dijalankan setiap kali method enqueue() dipanggil.
   - isEmpty(): method ini digunakan untuk memeriksa apakah antrian kosong atau tidak. Method ini mengembalikan true jika antrian kosong dan false jika antrian tidak kosong. Method ini memeriksa panjang antrian menggunakan properti length dari array pada antrian.

4.3 Penjelasan Tata Cara Penggunaan Program

Untuk menjalankan program, pengguna cukup menggunakan salah satu extension dari VS Code , yaitu : Live Server. Ketika ekstensi tersebut telah di-install, pengguna dapat langsung membuka direktori penyimpanan file program dan menjalankannya. Langkah-langkah untuk menjalankannya, antara lain:

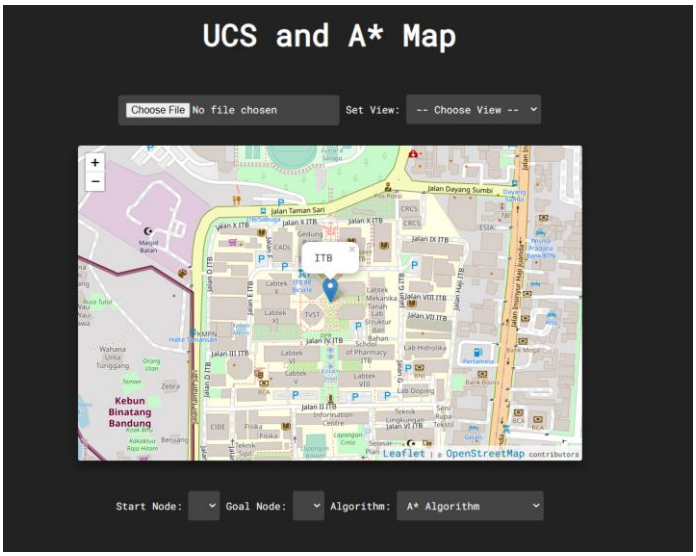
1. Buka file ‘index.html’ yang terletak pada folder ‘src’.
2. Klik logo ‘Go live’ yang tersedia di bagian bawah VS Code.



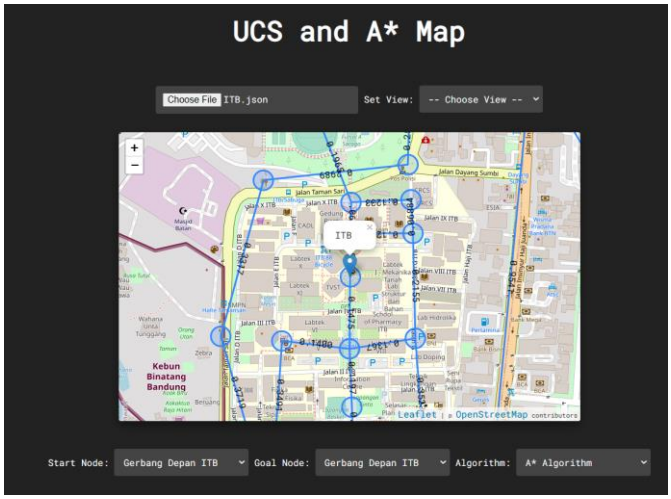
Gambar 4.3.1. *Tampilan Extensiom 'Live Server'*
Sumber: Dokumen Pribadi Penulis

Gambar 4.3.2. *Tampilan Tombol 'Go Live'*
Sumber: Dokumen Pribadi Penulis

3. Jika muncul dialog 'Select Default Browser', pilih browser yang akan digunakan untuk menjalankan program.
4. Setelah browser terbuka, pengguna dapat memilih file config yang ingin digunakan dengan memilih langsung via explorer. (file config telah tersedia pada folder 'test' yang ber-ekstensi *.json)
5. Sistem akan menampilkan rute terpendek dari titik awal ke titik tujuan dengan mengunakan algoritma A* atau UCS, tergantung dari opsi yang dipilih oleh pengguna pada menu dropdown.
6. Pengguna juga dapat mengganti opsi algoritma, zoom level pada peta, serta mengubah view yang akan ditampilkan di layar.
7. Cara alternatif untuk membuka program adalah melalui pranala berikut ini.



Gambar 4.3.3. *Tampilan Awal Program*
Sumber: Dokumen Pribadi Penulis



***Gambar 4.3.4**. Tampilan Peta oleh Program*
Sumber: Dokumen Pribadi Penulis

4.4 Hasil Pengujian

4.4.1. Config File yang digunakan


```
{
  "nodes": [
    {"name": "Gerbang Depan ITB", "lat": -6.893247, "lon": 107.610462},
    {"name": "Pintu Parkir SR", "lat": -6.893594, "lon": 107.61184},
    {"name": "Pintu Parkir Sipil", "lat": -6.893616, "lon": 107.60889},
    {"name": "Pertigaan Boromeus", "lat": -6.893791, "lon": 107.612935},
    {"name": "Lapbas Lapcin", "lat": -6.891704, "lon": 107.610408},
    {"name": "Labtek V/VIII", "lat": -6.890628, "lon": 107.610365},
    {"name": "GKUT", "lat": -6.890436, "lon": 107.611588},
    {"name": "GKUB", "lat": -6.890484, "lon": 107.609105},
    {"name": "Oktagon dll", "lat": -6.889302, "lon": 107.610381},
    {"name": "Sunken Court", "lat": -6.888498, "lon": 107.610381},
    {"name": "Gerbang tunnel", "lat": -6.887928, "lon": 107.610392},
    {"name": "Saraga", "lat": -6.886282, "lon": 107.609743},
    {"name": "Sabuga", "lat": -6.8860744776735405, "lon": 107.60822002437851},
    {"name": "CRCS", "lat": -6.88788, "lon": 107.611508},
    {"name": "Kebun Binatang", "lat": -6.890399, "lon": 107.607970},
    {"name": "TamFest", "lat": -6.887521470165671, "lon": 107.60876143113941},
    {"name": "Tamansari-Siliwangi", "lat": -6.884971, "lon": 107.611502},
    {"name": "Sangkuriang", "lat": -6.878618, "lon": 107.609974},
    {"name": "Taman Segitiga", "lat": -6.887232, "lon": 107.611453},
    {"name": "Simpang Dago", "lat": -6.885248, "lon": 107.613745},
    {"name": "ITHB", "lat": -6.889302794111865, "lon": 107.61610902313934},
    {"name": "Crisbar", "lat": -6.878922, "lon": 107.612618},
    {"name": "Borma Dago", "lat": -6.876882, "lon": 107.61773},
    {"name": "CAS", "lat": -6.888498, "lon": 107.611545}
  ],
```


***Gambar 4.4.1.1.** Sample Nodes ITB.json*
Sumber: Dokumen Pribadi Penulis


```
  "edges": [
    [
      0, 1, 1, 0, 1, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0
    ],
    [
      1, 0, 0, 1, 0, 0, 1, 0,
      0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0
    ],
    [
      1, 0, 0, 0, 0, 0, 0, 1,
      0, 0, 0, 0, 0, 0, 1, 0,
      0, 0, 0, 0, 0, 0, 0, 0
    ],
    [
      0, 1, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 1, 0, 0, 0, 0
    ],
    [
      1, 0, 0, 0, 0, 1, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0
    ],
```


***Gambar 4.4.1.2.** Sample Edges ITB.json*
Sumber: Dokumen Pribadi Penulis

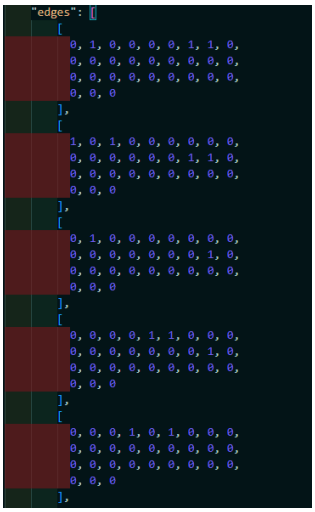

```
{
  "nodes": [
    {"name": "Masjid Agung Buah Batu", "lat": -6.954295260063078, "lon" : 107.63991356303526},
    {"name": "Pasar Kordon Kujangsari", "lat": -6.954223562353738, "lon":107.63917917247275},
    {"name": "Tugu Kordon", "lat": -6.954832352408868, "lon":107.64025060342464},
    {"name": "Pegadaian Pasar Kordon", "lat": -6.954980434293251, "lon": 107.6389825323762},
    {"name": "Pertigaan Gang Empang", "lat": -6.956996022375028, "lon": 107.63946793751319},
    {"name": "Stasiun Bandung", "lat": -6.958921940600545, "lon":107.63916743090337},
    {"name": "SPBU Buah Batu", "lat": -6.960483469967183, "lon": 107.63876211151195},
    {"name": "Aragon travel", "lat": -6.961857178066642, "lon": 107.63883831597404},
    {"name": "Praktek Spesialis Kulit Dan Kelamin", "lat": -6.962387572733704, "lon": 107.63840093585347},
    {"name": "Gerbang tol buah batu", "lat": -6.961318371476772, "lon": 107.63849631786441},
    {"name": "Masjid Baburrahman", "lat": -6.961036153139089, "lon": 107.6407878523069},
    {"name": "Kompleks Sapta Taruna (PU)", "lat": -6.965226670051706, "lon":107.63815767607245},
    {"name": "Transmart Buah Batu", "lat": -6.966796080870985, "lon": 107.63904699342923},
    {"name": "Puskesmas Bojongsoang", "lat": -6.967537217338871, "lon": 107.63731892302378},
    {"name": "Jalan Telekomunikasi", "lat": -6.972793256825605, "lon": 107.63597941173546},
    {"name": "Bundaran Tel U", "lat":-6.97244118903242, "lon": 107.63401789557321},
    {"name": "Tel U", "lat": -6.973064772163198, "lon": 107.63148114235497},
    {"name": "Masjid Syamsul Ulum Tel U", "lat": -6.975431190720805, "lon": 107.63212263824509},
    {"name": "Telkom University Landmark Tower (TULT)", "lat": -6.969301273947001, "lon": 107.62858004907605}
  ],
```


***Gambar 4.4.1.3.** Sample Nodes BuahBatu.json*
Sumber: Dokumen Pribadi Penulis

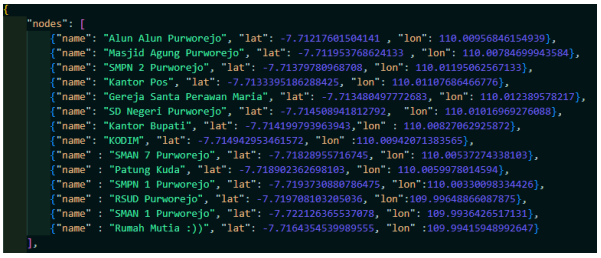

```
  "edges": [
    [
      0, 1, 1, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0
    ],
    [
      1, 0, 1, 1, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0
    ],
    [
      1, 1, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0
    ],
    [
      0, 1, 0, 0, 1, 0, 0,
      0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0
    ],
    [
      0, 0, 0, 1, 0, 1, 0,
      0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0
    ],
```


***Gambar 4.4.1.4.** Sample Nodes BuahBatu.json*
Sumber: Dokumen Pribadi Penulis

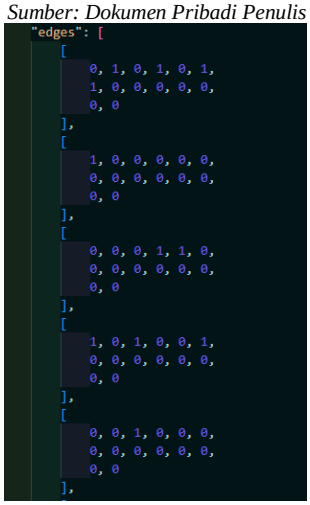
Gambar 4.4.1.5. *Sample Nodes Magelang.json*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.1.6. *Sample Nodes Magelang.json*
Sumber: Dokumen Pribadi Penulis



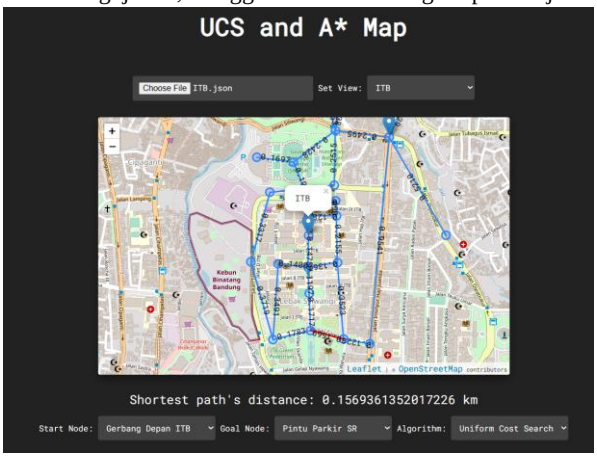
Gambar 4.4.1.7. *Sample Nodes Purworejo.json*
Sumber: Dokumen Pribadi Penulis



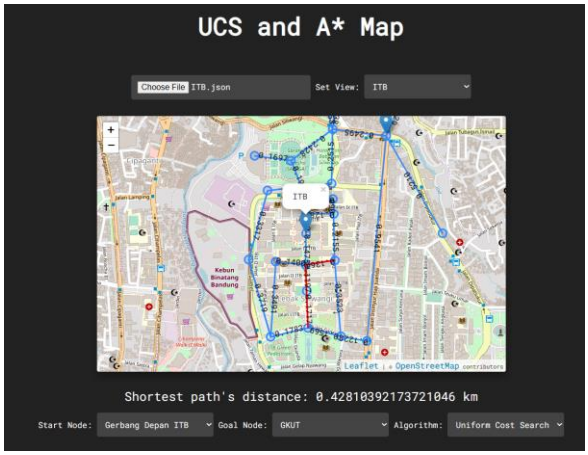
Gambar 4.4.1.8. *Sample Nodes Purworejo.json*
Sumber: Dokumen Pribadi Penulis

4.4.2. Algoritma Uniform Cost Search (UCS)

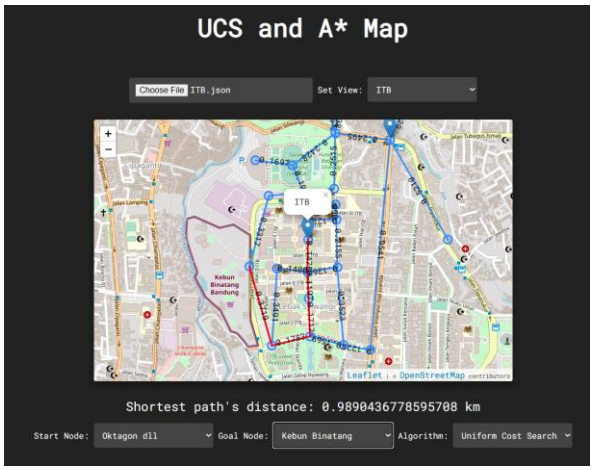
1. Pengujian 1, menggunakan File config Map : ITB.json



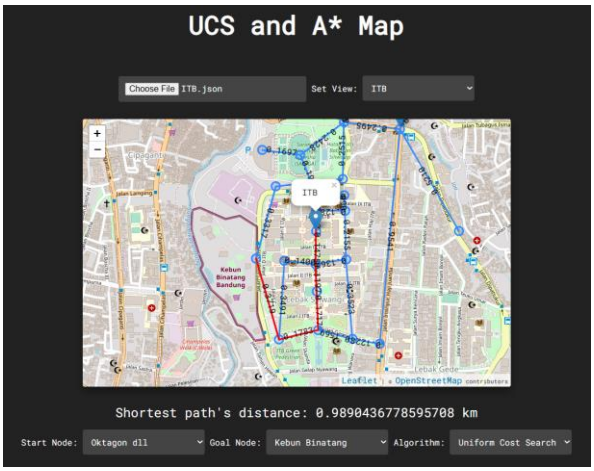
Gambar 4.4.2.1.1. *Gerbang Depan ITB ke Pintu Parkir SR*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.2.1.2. *Gerbang Depan ITB ke GKUT*
Sumber: Dokumen Pribadi Penulis

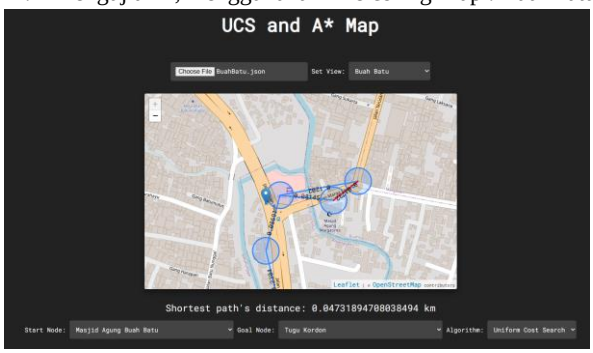


Gambar 4.4.2.1.3. *Oktagon ke Kebun Binatang*
Sumber: Dokumen Pribadi Penulis

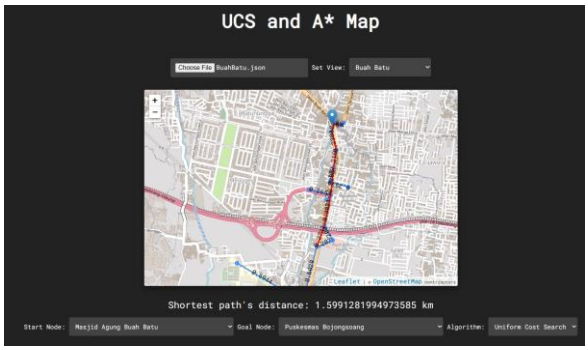


Gambar 4.4.2.1.4. *Oktagon ke TamFest*
Sumber: Dokumen Pribadi Penulis

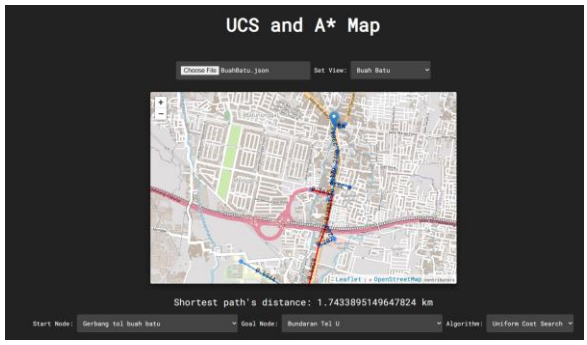
2. Pengujian 2, menggunakan File config Map : BuahBatu.json



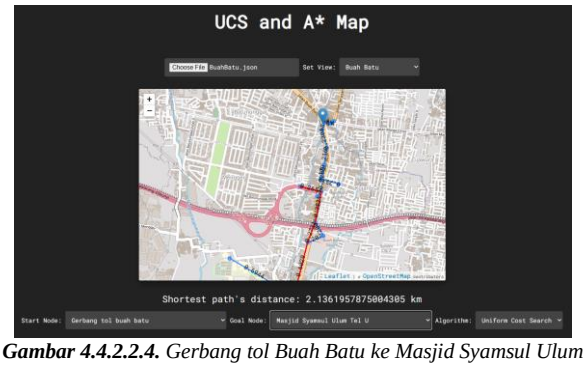
Gambar 4.4.2.2.1. *Masjid Agung Buah Batu ke Tugu Kordon*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.2.2.2. *Masjid Agung Buah Batu ke Puskesmas Bojongsoang*
Sumber: Dokumen Pribadi Penulis

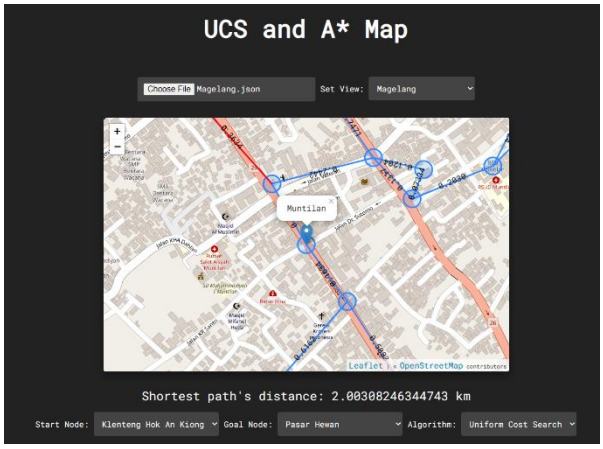


Gambar 4.4.2.2.3. *Gerbang tol Buah Batu ke Bundaran Tel U*
Sumber: Dokumen Pribadi Penulis

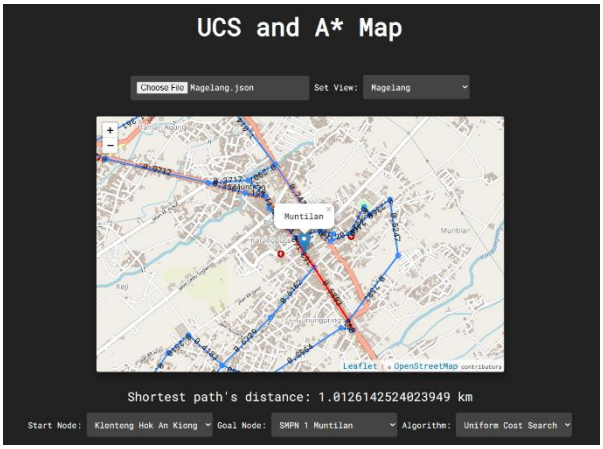


Gambar 4.4.2.2.4. *Gerbang tol Buah Batu ke Masjid Syamsul Ulum Tel U*
Sumber: Dokumen Pribadi Penulis

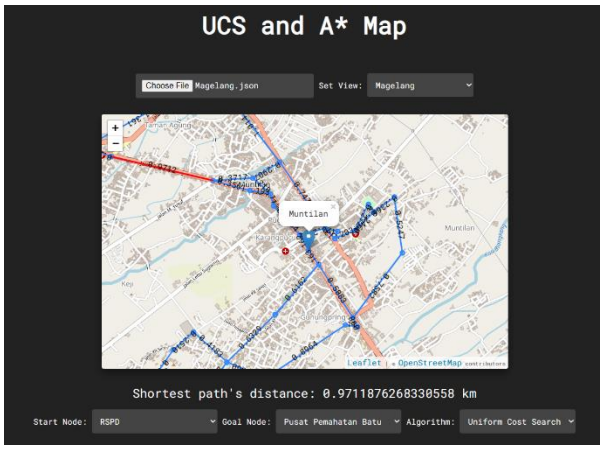
3. Pengujian 3, menggunakan File config Map : Magelang.json



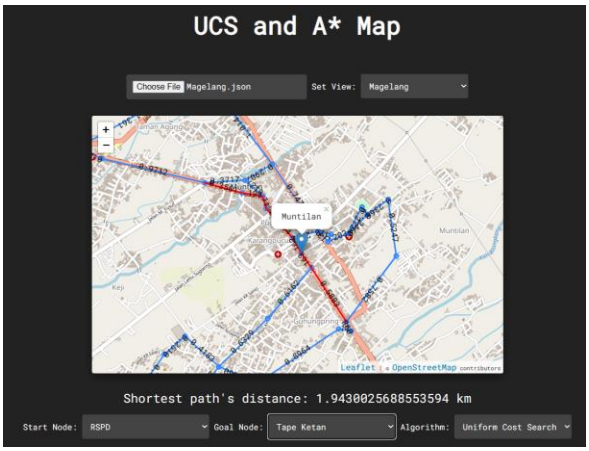
Gambar 4.4.2.3.1. *Klenteng Hok An Kiong ke Pasar Hewan*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.2.3.2. *Klenteng Hok An Kiong ke SMPN 1 Muntilan*
Sumber: Dokumen Pribadi Penulis

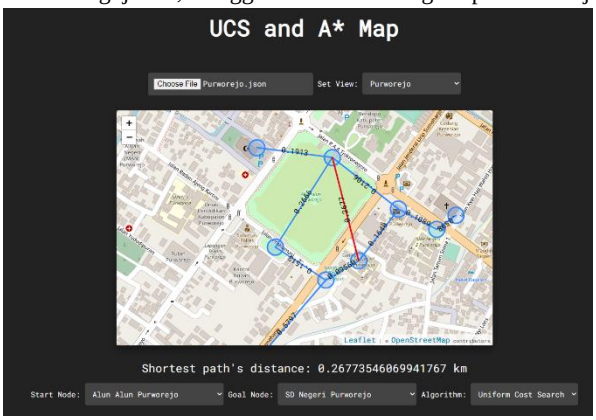


Gambar 4.4.2.3.3. *RSPD ke Pusat Pemahatan Batu*
Sumber: Dokumen Pribadi Penulis

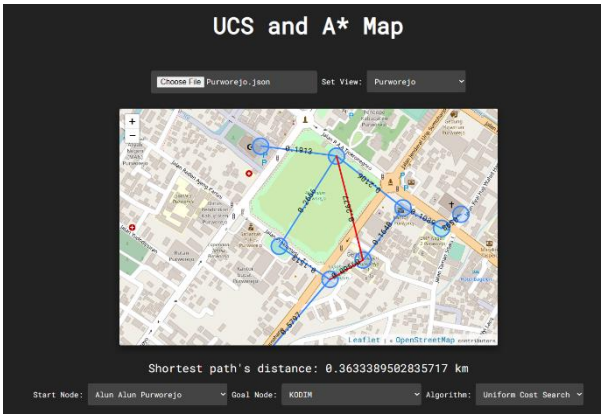


Gambar 4.4.2.3.4. *RSPD ke Tape Ketan*
Sumber: Dokumen Pribadi Penulis

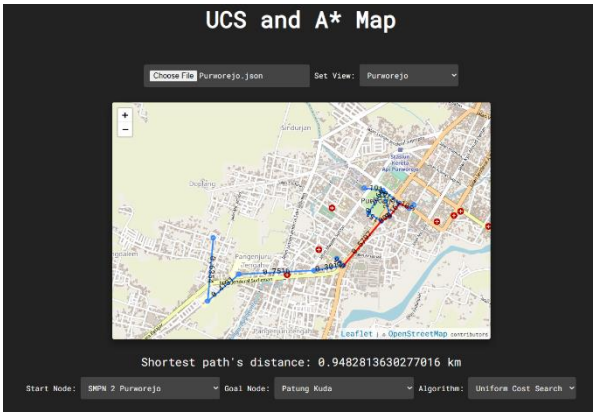
4. Pengujian 4, menggunakan File config Map : Purworejo.json



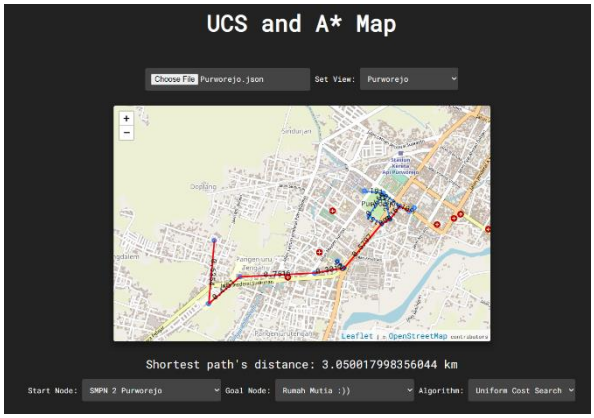
Gambar 4.4.2.4.1. *Alun Alun Purworejo ke SD Negeri Purworejo*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.2.4.2. *Alun Alun Purworejo ke KODIM*
Sumber: Dokumen Pribadi Penulis



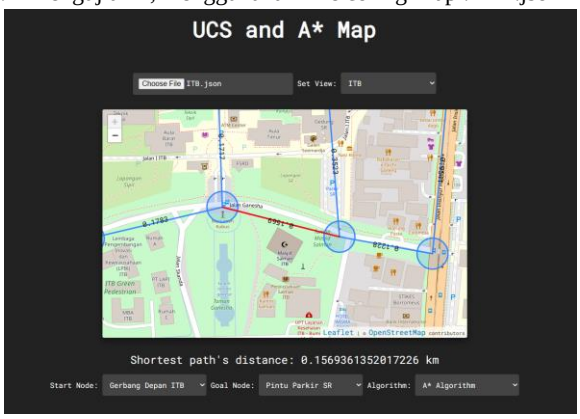
Gambar 4.4.2.4.3. *SMPN 2 Purworejo ke Patung Kuda*
Sumber: Dokumen Pribadi Penulis



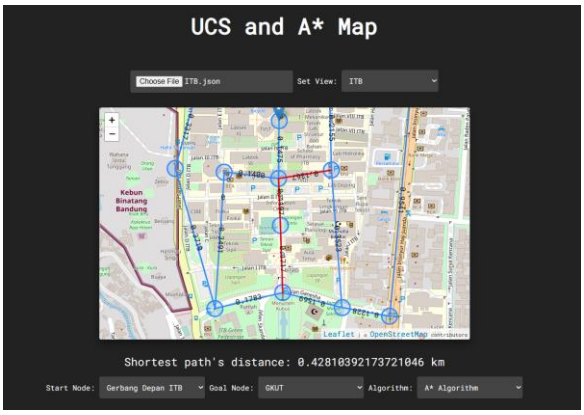
Gambar 4.4.2.4.4. *SMPN 2 Purworejo ke Rumah Mutia*
Sumber: Dokumen Pribadi Penulis

4.4.3. Algoritma A*

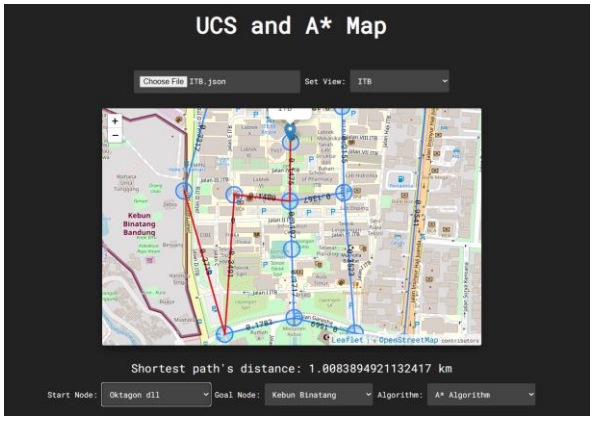
1. Pengujian 1, menggunakan File config Map : ITB.json



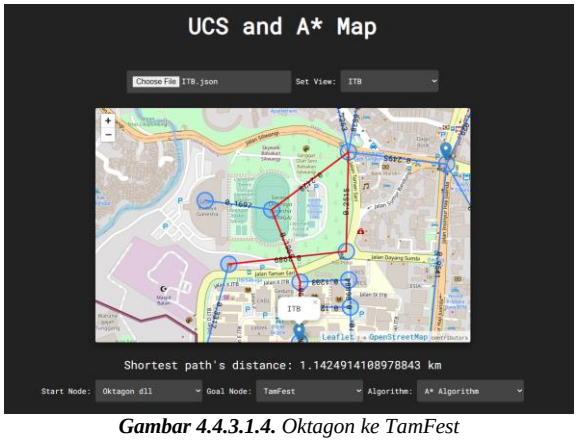
Gambar 4.4.3.1.1. *Gerbang Depan ITB ke Pintu Parkir SR*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.1.2. *Gerbang Depan ITB ke GKUT*
Sumber: Dokumen Pribadi Penulis

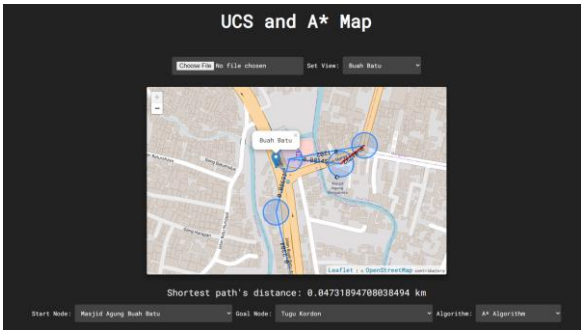


Gambar 4.4.3.1.3. *Oktagon ke Kebun Binatang*
Sumber: Dokumen Pribadi Penulis



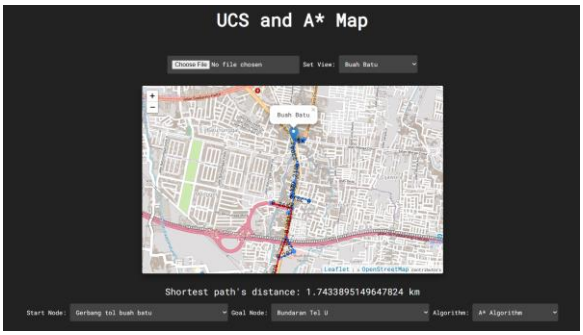
Gambar 4.4.3.1.4. *Oktagon ke TamFest*
Sumber: Dokumen Pribadi Penulis

2. Pengujian 2, menggunakan File config Map : BuahBatu.json

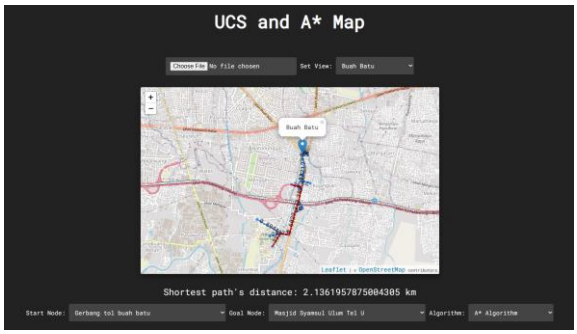


Gambar 4.4.3.2.1. *Masjid Agung Buah Batu ke Tugu Kordon*
Sumber: Dokumen Pribadi Penulis

Gambar 4.4.3.2.2. *Masjid Agung Buah Batu ke Puskesmas Bojongsoang*
Sumber: Dokumen Pribadi Penulis

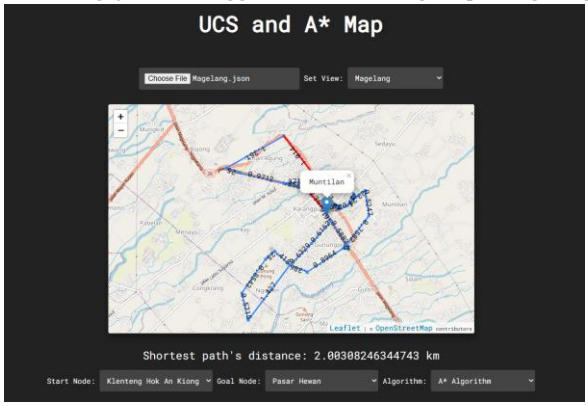


Gambar 4.4.3.2.3. *Gerbang tol Buah Batu ke Bundaran Tel U*
Sumber: Dokumen Pribadi Penulis

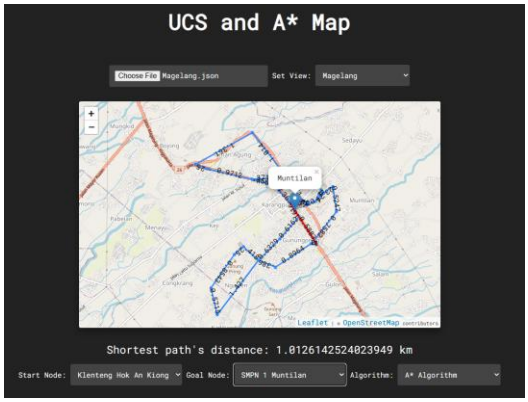


Gambar 4.4.3.2.4. *Gerbang tol Buah Batu ke Masjid Syamsul Ulum Tel U*
Sumber: Dokumen Pribadi Penulis

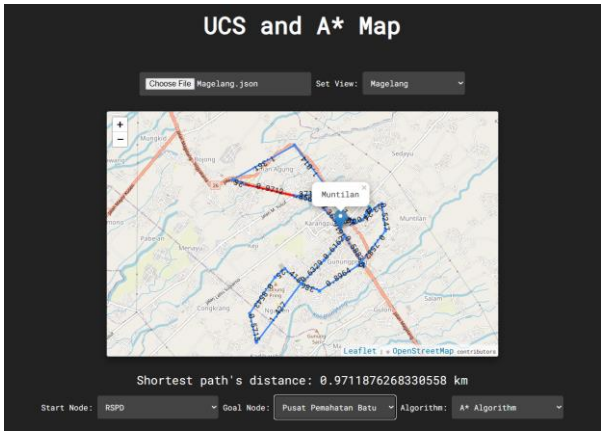
3. Pengujian 3, menggunakan File config Map : Magelang.json



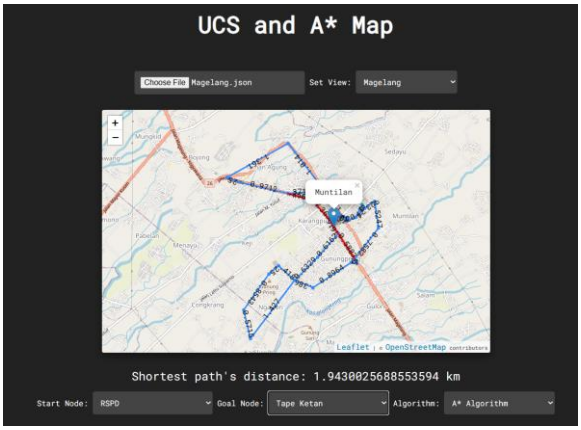
Gambar 4.4.3.3.1. *Klenteng Hok An Kiong ke Pasar Hewan*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.3.2. *Klenteng Hok An Kiong ke SMPN 1 Muntilan*
Sumber: Dokumen Pribadi Penulis

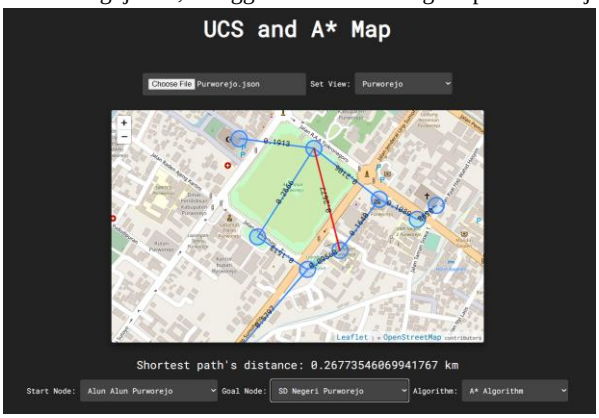


Gambar 4.4.3.3.3. *RSPD ke Pusat Pemahatan Batu*
Sumber: Dokumen Pribadi Penulis

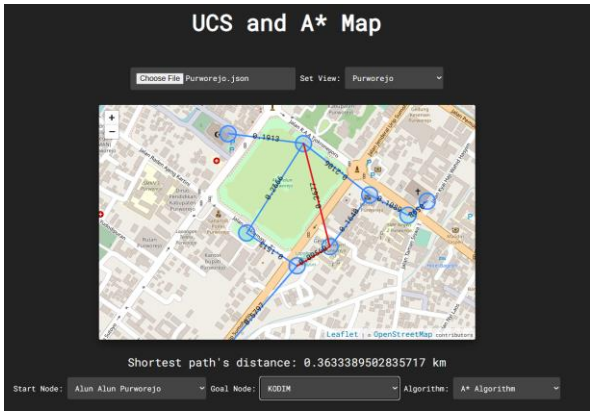


Gambar 4.4.3.3.4. *RSPD ke Tape Ketan*
Sumber: Dokumen Pribadi Penulis

4. Pengujian 4, menggunakan File config Map : Purworejo.json

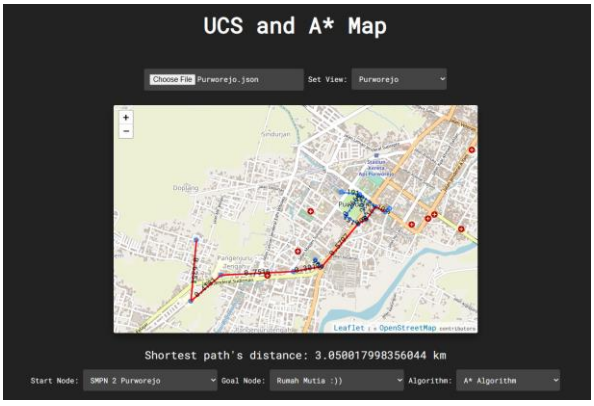


Gambar 4.4.3.4.1. *Alun Alun Purworejo ke SD Negeri Purworejo*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.4.2. *Alun Alun Purworejo ke KODIM*
Sumber: Dokumen Pribadi Penulis

Gambar 4.4.3.4.3. *SMPN 2 Purworejo ke Patung Kuda*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.4.4. *SMPN 2 Purworejo ke Rumah Mutia*
Sumber: Dokumen Pribadi Penulis

4.5 Analisis Desain Solusi Algoritma A* dan UCS

Algoritma UCS (Uniform Cost Search) dan A* adalah dua algoritma pencarian jalur yang paling sering digunakan dalam pemecahan permasalahan rute terpendek. UCS adalah algoritma yang bekerja dengan memilih node yang memiliki biaya terendah untuk diperluas. Algoritma ini tidak memperhitungkan heuristik atau informasi tentang tujuan akhir, dan hanya mengandalkan biaya dari setiap node yang

diperluas. Sedangkan A* adalah algoritma pencarian jalur yang menggabungkan heuristik dan biaya yang terkait dengan setiap node yang diperluas.

Untuk menganalisis perbandingan efektifitas dan efisiensi antara UCS dan A* dalam pemecahan permasalahan rute terpendek, kita perlu memperhatikan faktor-faktor berikut:

1. Kualitas heuristik yang digunakan: A* bekerja lebih efisien dan efektif jika heuristik yang digunakan memperhitungkan jarak langsung (euclidean distance) antara node dan tujuan akhir. Jika heuristik yang digunakan buruk atau tidak akurat, maka kinerja A* dapat menjadi lebih buruk dari UCS.
2. Ukuran peta dan jumlah node: Pada peta yang besar dan memiliki banyak node, A* dapat lebih efektif dan efisien karena mempertimbangkan heuristik yang baik dapat membantu algoritma menghindari memperluas node yang tidak perlu, sehingga mengurangi jumlah node yang harus diperluas.

3. Kondisi topologi peta: Pada peta yang memiliki topologi yang kompleks atau banyak rintangan, UCS dapat menjadi lebih efektif karena algoritma ini tidak mempertimbangkan heuristik dan hanya memperhitungkan biaya dari setiap node yang diperluas. Sebaliknya, A* dapat menghabiskan banyak waktu mempertimbangkan heuristik untuk mencari jalan keluar dari rintangan.

Dari analisis di atas, dapat disimpulkan bahwa A* cenderung lebih efektif dan efisien dibandingkan UCS dalam menyelesaikan permasalahan rute terpendek jika heuristik yang digunakan akurat, pada peta yang besar dan kompleks. Namun, jika heuristik buruk atau peta memiliki topologi yang sederhana, UCS dapat menjadi pilihan yang lebih baik karena sifatnya yang hanya mempertimbangkan biaya dari setiap node yang diperluas.

Oleh karena peta yang digunakan cukup sederhana dan heuristik A* yang kurang begitu baik, maka pada kasus tugas kecil kali ini, algoritma UCS memiliki tingkat keakuratan yang cenderung lebih baik. Hal tersebut terbukti dari hasil pengujian pada bab 4.4 yang menunjukkan bahwa pada kasus tertentu, nilai rute terpendek yang dihasilkan algoritma UCS lebih kecil atau sama dengan algoritma A*. Fakta tersebut sesuai dengan teori yang mengatakan bahwa pencarian rute terpendek dengan algoritma UCS selalu menghasilkan nilai yang optimal.

BAB V
Checklist dan Pranala Terkait

5.1 Checklist

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	√	

5.2 Pranala Terkait

Link Repository:
https://github.com/Ulung32/Tucil3_13521122_13521166

BAB VI
Kesimpulan Saran dan Refleksi

6.1 Kesimpulan

Dari tugas kecil IF2211 Strategi Algoritma semester 2 2022/2023 berjudul "Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek", kami berhasil membuat Aplikasi berbasis Web yang dapat melakukan pencarian rute terpendek dari satu titik ke titik lain menggunakan algoritma A* dan Uniform Cost Search (UCS).

Kami juga telah berhasil memanfaatkan bahasa pemrogram HTML untuk mendesain kerangka Web, CSS untuk mempercantik tampilan kerangka yang telah dibuat, serta JavaScript untuk menulis kode program yang digunakan. Web akan menerima masukkan file config (*.json) dari user, dilanjutkan dengan user memilih view yang ingin ditampilkan, start node dan goal node, serta algoritma yang dipilih. Web kemudian akan menampilkan rute terpendek antar kedua titik.

Kesimpulan yang kami dapatkan dari pengerjaan tugas kecil ini, antara lain:

- Algoritma A* dan Uniform Cost Search dapat digunakan untuk menyelesaikan permasalahan pencarian rute terpendek. Algoritma A* melakukan penelusuran rute dengan metode pencarian heuristik, sedangkan algoritma UCS melakukan pencarian dengan metode uninformed search.

6.2 Saran

Pengembangan Web application memerlukan waktu yang tidak sebentar, belum lagi pencarian API gratis yang dapat digunakan untuk menampilkan peta. Oleh karena itu, untuk kedepannya, sebaiknya jika hendak menggunakan Web based platform, ditambah waktu pengerjaannya. Selain itu, berikan pula referensi untuk mendapatkan API secara gratis. Selain itu, spesifikasi tugas sebaiknya lebih diperjelas dan pastikan tidak ada ambiguitas lagi.

6.3 Refleksi

Refleksi dari kelompok kami mungkin terkait komunikasi dan sistem pengerjaan yang dilakukan selama rentang waktu tugas besar ini. Dari segi komunikasi, mungkin seharusnya dilakukan lebih sering secara offline atau pertemuan terjadwal, agar komunikasi dapat terjalin dengan lebih baik lagi.

6.4 Tanggapan

Tugas Kecil 3 IF2211 – Strategi Algoritma 2022/2023 cukup menantang. Hal ini disebabkan oleh kami yang dipacu untuk membuat sebuah Web Based Application, mencari API peta, serta merancang algoritma A* dan UCS dalam waktu yang singkat.

Referensi

Munir, Rinaldi, Nur Ulfa Mauladevi. 2023, “Diktat Penentuan rute (Route/Path Planning)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf, diakses pada 8 April 2023.

LeafletJS. 2023, “Leaflet – a JavaScript library for interactivve maps”, https://leafletjs.com/, diakses pada 8 April 2023.

OpenStreetMap. 2023, “OpenStreetMap Documentation”, https://www.openstreetmap.org, diakses pada 8 April 2023.

GeeksForGeeks. 2023, “OpenStreetMap Documentation”, https://www.openstreetmap.org, diakses pada 8 April 2023.

GeeksForGeeks. 2023, “A* Search Algorithm”, https://www.geeksforgeeks.org/a-search-algorithm/, diakses pada 8 April 2023.