

# **LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA**

## **IMPLEMENTASI ALGORITMA UCS DAN A\* UNTUK MENENTUKAN LINTASAN TERPENDEK**

**Dosen : Dr. Ir. Rinaldi, M.T.**



**Disusun oleh :**

**Vieri Fajar Firdaus      13521099**

**Saddam Annais S      13521121**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2022/2023**

## **Daftar Isi**

<b>Daftar Isi.....</b>	<b>1</b>
<b>DESKRIPSI MASALAH.....</b>	<b>2</b>
<b>ALGORITMA.....</b>	<b>3</b>
<b>SOURCE CODE.....</b>	<b>7</b>
<b>TEST CASE.....</b>	<b>13</b>
<b>TABLE.....</b>	<b>17</b>
<b>Kesimpulan Saran dan Refleksi.....</b>	<b>18</b>
<b>Lampiran.....</b>	<b>19</b>

## 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.

Spesifikasi program:

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.

## ALGORITMA

Landasan teori yang akan digunakan pada penerapan strategi algoritma menentukan lintasan terpendek yaitu : algoritma *UCS* dan algoritma *A\**. Sedangkan untuk tampilannya menggunakan menggunakan framework react

### A. Algoritma UCS

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian jalur terpendek yang mempertimbangkan biaya setiap langkah menuju tujuan. Algoritma ini bekerja dengan menjelajahi graf dari simpul awal ke simpul tujuan, mencari jalur dengan biaya terendah.

Algoritma UCS bekerja dengan menggunakan sebuah priority queue atau antrian prioritas yang diurutkan berdasarkan biaya terkecil. Pada setiap tahap, simpul dengan biaya terkecil yang belum dieksplorasi akan dipilih untuk diperiksa lebih lanjut. Jika simpul tersebut adalah simpul tujuan, maka algoritma akan mengembalikan jalur terpendek yang ditemukan. Jika simpul tersebut bukan simpul tujuan, maka algoritma akan mengeksplorasi simpul-simpul tetangga dari simpul tersebut dan memperbarui cost atau biaya dari setiap simpul tetangga.

Kelebihan dari algoritma UCS adalah mampu menemukan jalur terpendek dengan biaya yang optimal, namun kekurangannya adalah algoritma ini membutuhkan waktu yang cukup lama untuk menjelajahi graf yang besar atau rumit. Oleh karena itu, algoritma ini lebih cocok digunakan pada graf yang relatif kecil atau pada kasus yang tidak terlalu kompleks.

Algoritma UCS (Uniform Cost Search) bekerja dengan mengikuti langkah-langkah berikut:

1. Inisialisasi: Tentukan simpul awal, simpul tujuan, dan biaya awal (cost) dari simpul awal ke simpul awal adalah 0. Simpan simpul awal pada antrian prioritas dengan biaya 0.
2. Ambil simpul dengan biaya terkecil: Ambil simpul dengan biaya terkecil dari antrian prioritas. Jika simpul ini adalah simpul tujuan, maka algoritma akan mengembalikan jalur terpendek yang ditemukan. Jika tidak, maka pindah ke langkah berikutnya.
3. Eksplorasi simpul: Eksplorasi simpul yang dipilih pada langkah sebelumnya dan periksa simpul-simpul tetangga dari simpul tersebut. Jika simpul tetangga belum dieksplorasi, maka tambahkan simpul tersebut ke

antrian prioritas dengan biaya yang sesuai (biaya dari simpul awal ke simpul tetangga). Jika simpul tetangga sudah dieksplorasi dan biaya dari simpul awal ke simpul tetangga yang baru lebih kecil dari biaya yang sebelumnya, maka perbarui biaya simpul tetangga dan simpan jalur terpendek yang ditemukan.

4. Ulangi langkah dua dan tiga sampai simpul tujuan ditemukan atau antrian prioritas kosong (tidak ada jalur yang dapat diambil lagi).
5. Jika simpul tujuan ditemukan, kembalikan jalur terpendek yang ditemukan dengan biaya totalnya. Jika tidak, kembalikan pesan bahwa tidak ada jalur yang ditemukan.

Dengan cara ini, algoritma UCS dapat menemukan jalur terpendek dari simpul awal ke simpul tujuan dengan mempertimbangkan biaya setiap langkah. Namun, algoritma ini dapat memakan waktu yang cukup lama jika digunakan pada graf yang besar atau rumit.

#### B. Algoritma A\*

Algoritma A\* adalah algoritma pencarian jalur terpendek yang mempertimbangkan biaya setiap langkah menuju tujuan serta estimasi jarak yang tersisa (heuristik). Algoritma ini memilih jalur dengan biaya terkecil yang juga memiliki estimasi jarak tersisa yang paling kecil menuju tujuan.

Algoritma A\* bekerja dengan cara mengeksplorasi simpul-simpul dari simpul awal ke simpul tujuan dengan cara menghitung biaya dari simpul awal ke simpul tersebut ( $g$ ), dan estimasi jarak tersisa dari simpul tersebut ke simpul tujuan ( $h$ ). Biaya total jalur ( $f$ ) dihitung dengan menjumlahkan biaya  $g$  dan estimasi jarak tersisa  $h$ .

Untuk menjalankan algoritma A\*, diperlukan sebuah heuristik yang dapat memberikan estimasi jarak tersisa dari simpul yang sedang dieksplorasi ke simpul tujuan. Heuristik ini harus adil, artinya tidak boleh memberikan estimasi yang terlalu besar atau terlalu kecil, sehingga algoritma dapat menghasilkan jalur terpendek dengan biaya yang optimal.

Kelebihan dari algoritma A\* adalah dapat menemukan jalur terpendek dengan biaya yang optimal dengan mempertimbangkan heuristik, sehingga algoritma ini biasanya lebih efisien dibandingkan dengan algoritma UCS. Namun, kelemahan dari algoritma A\* adalah algoritma ini memerlukan heuristik yang adil dan tidak

selalu dapat menemukan jalur terpendek jika heuristik yang digunakan tidak cukup akurat.

Dalam praktiknya, algoritma A\* banyak digunakan dalam aplikasi game, robotika, dan navigasi, serta dalam masalah optimisasi dan perencanaan rute.

Algoritma A\* bekerja dengan mengikuti langkah-langkah berikut:

1. Inisialisasi: Tentukan simpul awal, simpul tujuan, dan fungsi heuristik ( $h$ ) yang memberikan estimasi jarak tersisa dari simpul yang sedang diproses ke simpul tujuan. Set biaya awal (cost) dari simpul awal ke simpul awal menjadi 0. Simpan simpul awal pada antrian prioritas dengan biaya 0 dan fungsi heuristik  $h$ .
2. Ambil simpul dengan  $f$  terkecil: Ambil simpul dengan  $f$  terkecil dari antrian prioritas. Jika simpul ini adalah simpul tujuan, maka algoritma akan mengembalikan jalur terpendek yang ditemukan. Jika tidak, maka pindah ke langkah berikutnya.
3. Eksplorasi simpul: Eksplorasi simpul yang dipilih pada langkah sebelumnya dan periksa simpul-simpul tetangga dari simpul tersebut. Untuk setiap simpul tetangga, hitung biaya total jalur ( $f$ ) dengan menjumlahkan biaya  $g$  dari simpul awal ke simpul tersebut dengan estimasi jarak tersisa  $h$  dari simpul tersebut ke simpul tujuan. Jika simpul tetangga belum dieksplorasi atau biaya total jalur yang baru lebih kecil dari biaya total jalur yang sebelumnya, tambahkan simpul tersebut ke antrian prioritas dengan biaya total jalur yang baru.
4. Ulangi langkah dua dan tiga sampai simpul tujuan ditemukan atau antrian prioritas kosong (tidak ada jalur yang dapat diambil lagi).
5. Jika simpul tujuan ditemukan, kembalikan jalur terpendek yang ditemukan dengan biaya totalnya. Jika tidak, kembalikan pesan bahwa tidak ada jalur yang ditemukan.

Dengan cara ini, algoritma A\* dapat menemukan jalur terpendek dari simpul awal ke simpul tujuan dengan mempertimbangkan biaya setiap langkah serta estimasi jarak tersisa menggunakan fungsi heuristik. Algoritma ini biasanya lebih efisien dibandingkan dengan algoritma UCS, namun memerlukan heuristik yang adil dan

tidak selalu dapat menemukan jalur terpendek jika heuristik yang digunakan tidak cukup akurat.

### C. React

React adalah sebuah library JavaScript yang digunakan untuk membangun antarmuka pengguna (user interface/UI) pada aplikasi web. React dikembangkan oleh Facebook dan dirilis secara open source pada tahun 2013. React menggunakan konsep komponen (component-based) yang memungkinkan pengembang untuk membagi program menjadi bagian-bagian yang lebih kecil dan terorganisir dengan baik.

Salah satu keunggulan React adalah kemampuannya untuk memperbarui tampilan (render) secara efisien dan cepat. Hal ini karena React menggunakan Virtual DOM (Document Object Model) yang memungkinkan untuk memperbarui tampilan hanya pada bagian yang berubah, tanpa harus memperbarui seluruh halaman. Dengan cara ini, aplikasi yang dibangun dapat memberikan kesan yang lebih responsif dan cepat.

Berikut merupakan langkah-langkah untuk membuat program menggunakan React

1. Install Node.js: Node.js diperlukan dalam menjalankan React.
2. Buat proyek React baru: Setelah Node.js terinstal, buat proyek baru dengan menjalankan perintah `npx create-react-app <nama-proyek>` pada terminal. Ini akan membuat struktur proyek React dasar.
3. Mulai mengembangkan aplikasi: Setelah proyek dibuat, komponen dan logika dapat dibuat dan ditambahkan dalam folder `src`.
4. Jalankan aplikasi: Jalankan perintah `npm start` pada terminal. Aplikasi akan berjalan pada `http://localhost:3000` di browser dan setiap perubahan pada kode akan otomatis dimuat ulang pada browser.
5. Deploy aplikasi Anda: Setelah selesai mengembangkan aplikasi, aplikasi dapat dideploynya ke server web atau dengan menggunakan platform cloud seperti Heroku atau Netlify.

## SOURCE CODE

### A. Algoritma A\*

```
class PriorityQueue{
    constructor() {
        this.elements = [];
    }

    enqueue(element,priority) {
        const newNode = {element,priority};
        let added = false;

        for(let i=0;i<this.elements.length;i++) {
            if(priority < this.elements[i].priority) {
                this.elements.splice(i,0,newNode);
                added = true;
                break;
            }
        }

        if(!added) {
            this.elements.push(newNode);
        }
    }

    dequeue() {
        return this.elements.shift();
    }

    isEmpty() {
        return this.elements.length === 0;
    }
}

function distance(a,b,keterangan) {

    // map biasa
    if(keterangan) {
```

```

        return Math.sqrt((a[0]-b[0])**2 + (a[1]-b[1])**2)
    }
    // map peta
} else{
    return distance_map(a[0],a[1],b[0],b[1])
}
}

function distance_map(lat1, lon1, lat2, lon2) {
    const R = 6371000; // approximate radius of Earth in meters
    const dLat = (lat2 - lat1) * Math.PI / 180;
    const dLon = (lon2 - lon1) * Math.PI / 180;
    const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
              Math.cos(lat1 * Math.PI / 180) * Math.cos(lat2 *
Math.PI / 180) *
              Math.sin(dLon / 2) * Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    const distance = R * c;
    return distance/1000;
}

function aStar(fullGraph, start, finish, keterangan){
    let graph = fullGraph.matrix;
    let point = fullGraph.coordinates;
    const liveNode = new PriorityQueue()
    for(let i=0;i<graph.length;i++){
        for(let j=0;j<graph.length;j++) {
            if(graph[i][j] !=0) {
                graph[i][j]
                =distance(point[i],point[j],keterangan);
            }
        }
    }
    liveNode.enqueue([start],0)
    let pathTotal=null
    const weight = new Array(graph.length).fill(Infinity)
    weight[start] = 0
    if(start<0 || finish<0 || start>=graph.length ||
    finish>=graph.length || point.length!=graph.length){

```

```

        pathTotal=null
        return {pathTotal,weight}
    }
    while(!liveNode.isEmpty()){
        // console.log(liveNode.elements)
        const current = liveNode.dequeue()
        if(current.element[0] === finish){
            pathTotal = current.element
            break;
        }

        for(let i=0;i<graph[current.element[0]].length;i++){
            if(graph[current.element[0]][i] > 0){
                const newDistance = weight[current.element[0]] +
graph[current.element[0]][i]
                // console.log(current.element[0],i)
                if(newDistance < weight[i]){
                    const straightLineDistance =
distance(point[i],point[finish],keterangan)
                    const totalDistance= newDistance +
straightLineDistance
                    weight[i] = newDistance
                    const newNode= [i].concat(current.element)
                    liveNode.enqueue(newNode,totalDistance)
                }
            }
        }
    }

    let distanceMinimum=weight[finish]
    return {pathTotal,distanceMinimum}
}

function parserInputA(inputStr,keterangan) {
    const lines = inputStr.trim().split('\n');
    const matrix = [];

    let n = lines[0].trim().split(' ').length
    for (let i = 0; i < n; i++) {
        const row = lines[i].trim().split(' ').map(Number);
        matrix.push(row);
    }
    return matrix;
}

```

```

        matrix.push(row);
    }

const coordinates = [];
for (let i = n+1; i < lines.length; i++) {
    const coord = lines[i].trim().split(' ').map(Number);
    coordinates.push(coord);
}

for(let i=0;i<coordinates.length;i++){
    for(let j=0;j<coordinates.length;j++) {
        if(matrix[i][j] != 0){
            matrix[i][j] =
distance(coordinates[i],coordinates[j],keterangan)
        }
    }
}
return {matrix, coordinates};
}

export {aStar, parserInputA, distance,distance_map};

```

## B. Algoritma UCS

```

class PriorityQueue{
constructor() {
    this.elements = [];
}

enqueue(element,priority){
    const newNode = {element,priority};
    let added = false;

    for(let i=0;i<this.elements.length;i++){
        if(priority < this.elements[i].priority){
            this.elements.splice(i,0,newNode);
            added = true;
            break;
        }
    }
}

```

```

        if (!added) {
            this.elements.push(newNode);
        }
    }

dequeue() {
    return this.elements.shift();
}

isEmpty() {
    return this.elements.length === 0;
}
}

function UCS(graph, start, finish) {
    const liveNode = new PriorityQueue()

    liveNode.enqueue([start], 0)
    let pathTotal=null

    if(start<0 || finish<0 || start>=graph.length || finish>=graph.length){
        pathTotal=null
    }

    const weight = new Array(graph.length).fill(Infinity)
    weight[start] = 0

    // console.log("livenode")
    // console.log(liveNode.dequeue())

    while(!liveNode.isEmpty()){
        const current = liveNode.dequeue()
        // console.log("current",current)
        if(current.element[0] === finish){
            pathTotal = current.element
            break;
        }
    }
}

```

```

        for(let i=0;i<graph[current.element[0]].length;i++) {
            if(graph[current.element[0]][i] > 0){
                const newWeight = current.priority +
graph[current.element[0]][i]
                if(newWeight < weight[i]){
                    weight[i] = newWeight
                    const priority = newWeight
                    const newNode= [i].concat(current.element)
                    liveNode.enqueue(newNode,priority)
                }
            }
        }
    }

    let distanceMinimum=weight[finish]

    return {pathTotal,distanceMinimum}
}

function parserInputUCS(graph) {
    const rows = graph.trim().split("\n");
    const matrix = rows.map((row) => row.split(" ")).map((val) =>
parseInt(val)));
    return matrix;
}

```

### C. Algoritma CheckInput

```

function checkInput(inputString, keterangan) {
    // Split the input string into lines
    const lines = inputString.trim().split("\n");

    // menghapus baris kosong
    for (let i = 0; i < lines.length; i++) {

```

```

if (lines[i] === "" || lines[i] === "\r") {
    lines.splice(i, 1);
}
}

const rows = lines[0].trim().split(" ").length;
const matrix = [];

for (let i = 0; i < rows; i++) {
    matrix.push(lines[i].trim().split(" "));
}

// mengecek baris dan kolom harus sama
for (let i = 0; i < rows; i++) {
    if (matrix[i].length !== rows) {
        return false;
    }
}

// elemen 1/0
for (let i = 0; i < rows; i++) {
    for (let j = 0; j < rows; j++) {
        if (matrix[i][j] !== 1 && matrix[i][j] !== 0) {
            return false;
        }
    }
}

// mengecek (x,y) = (y,x)
for (let i = 0; i < rows; i++) {
    for (let j = 0; j < rows; j++) {
        if (matrix[i][j] !== matrix[j][i]) {
            return false;
        }
    }
}

// kalau jumlah point dan row beda
if (rows * 2 !== lines.length) {

```

```
    return false;
}

const point = [];
for (let i = rows; i < lines.length; i++) {
    point.push(lines[i].trim().split(" "));
}

// kalau banyak titik tidak 2
for (let i = 0; i < point.length; i++) {
    if (point[i].length !== 2) {
        return false;
    }
}

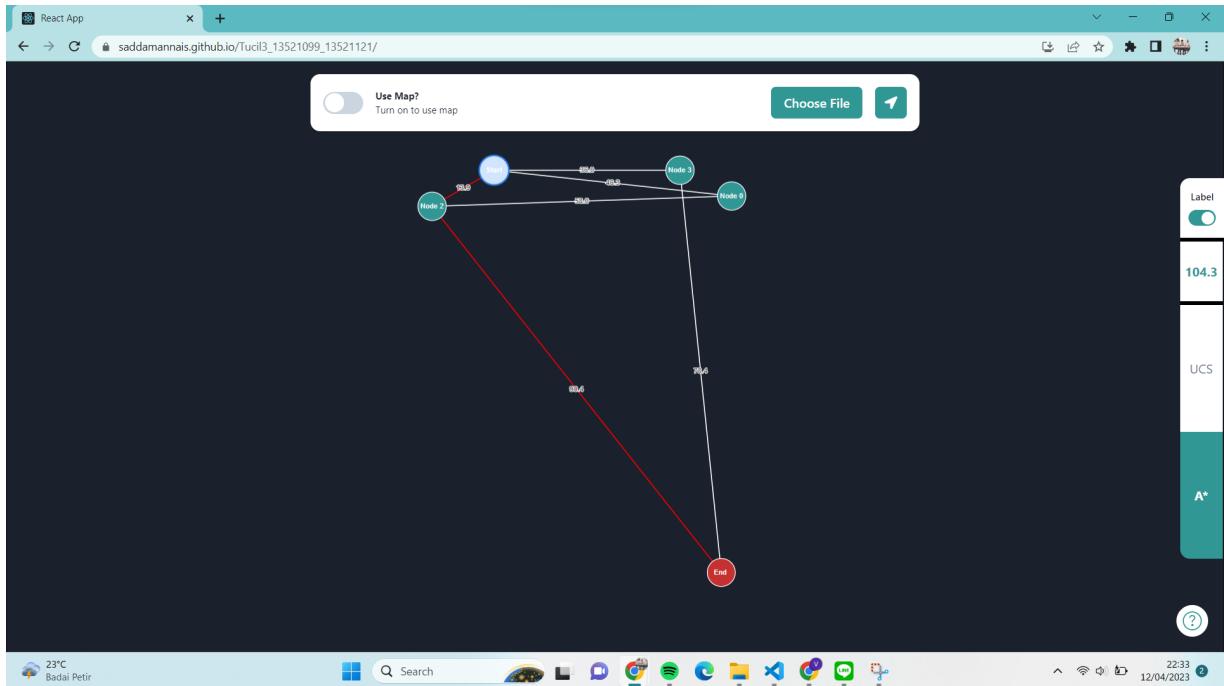
if (!keterangan) {
    for (let i = 0; i < point.length; i++) {
        if (Math.abs(point[i][0]) > 180 || Math.abs(point[i][1]) > 180) {
            return false;
        }
    }
}
return true;
}

export default checkInput;
```

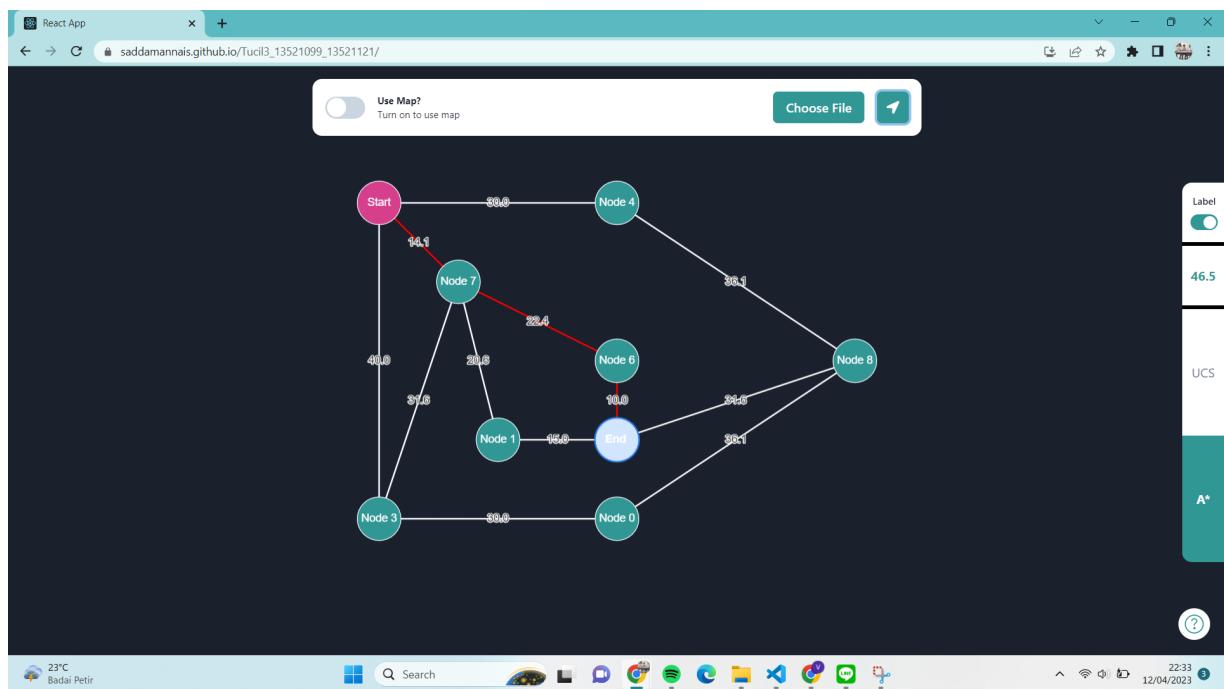
# TEST CASE

## A. Spesifikasi Wajib

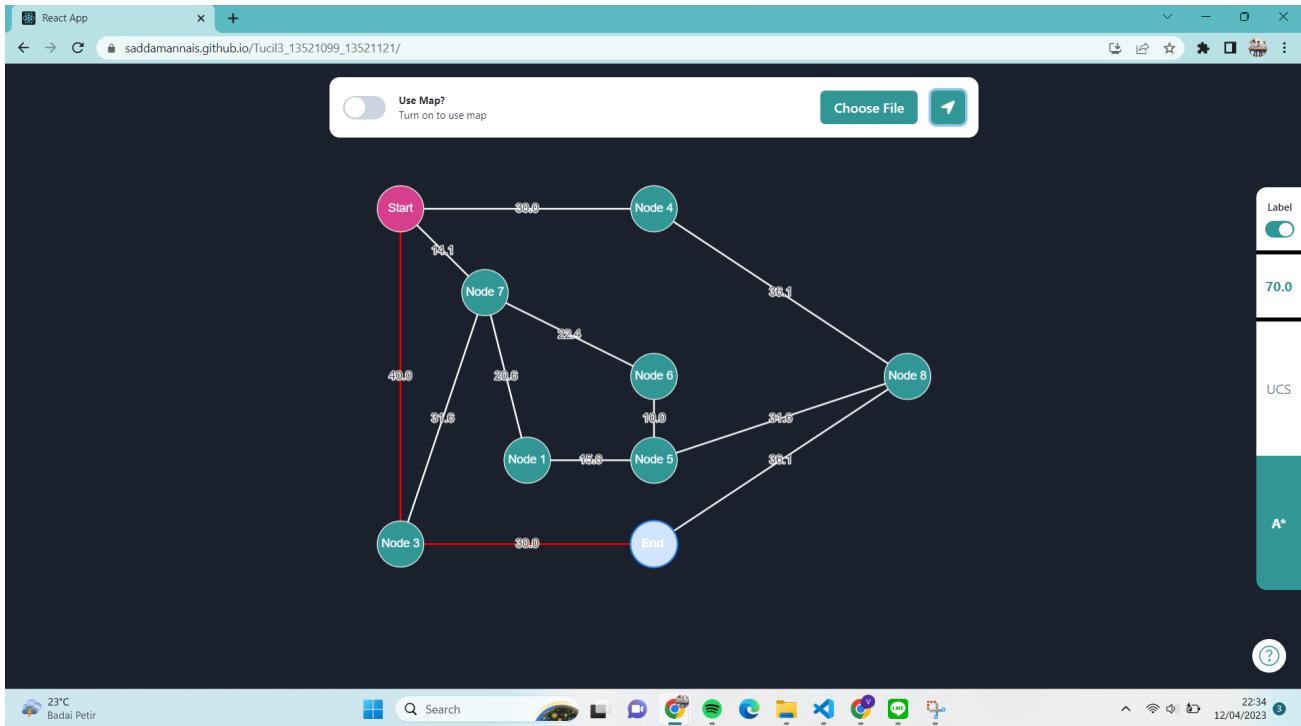
### 1. Testcase 1



### 2. Testcase 2

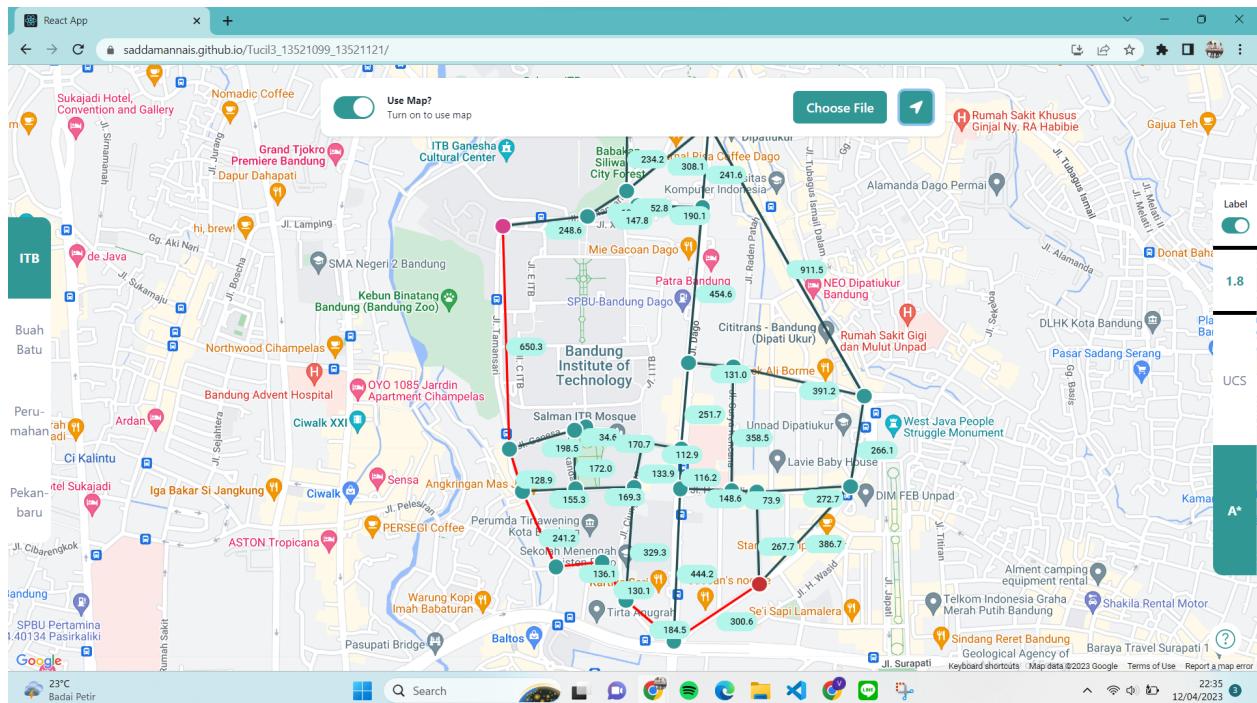


### 3. Testcase 3

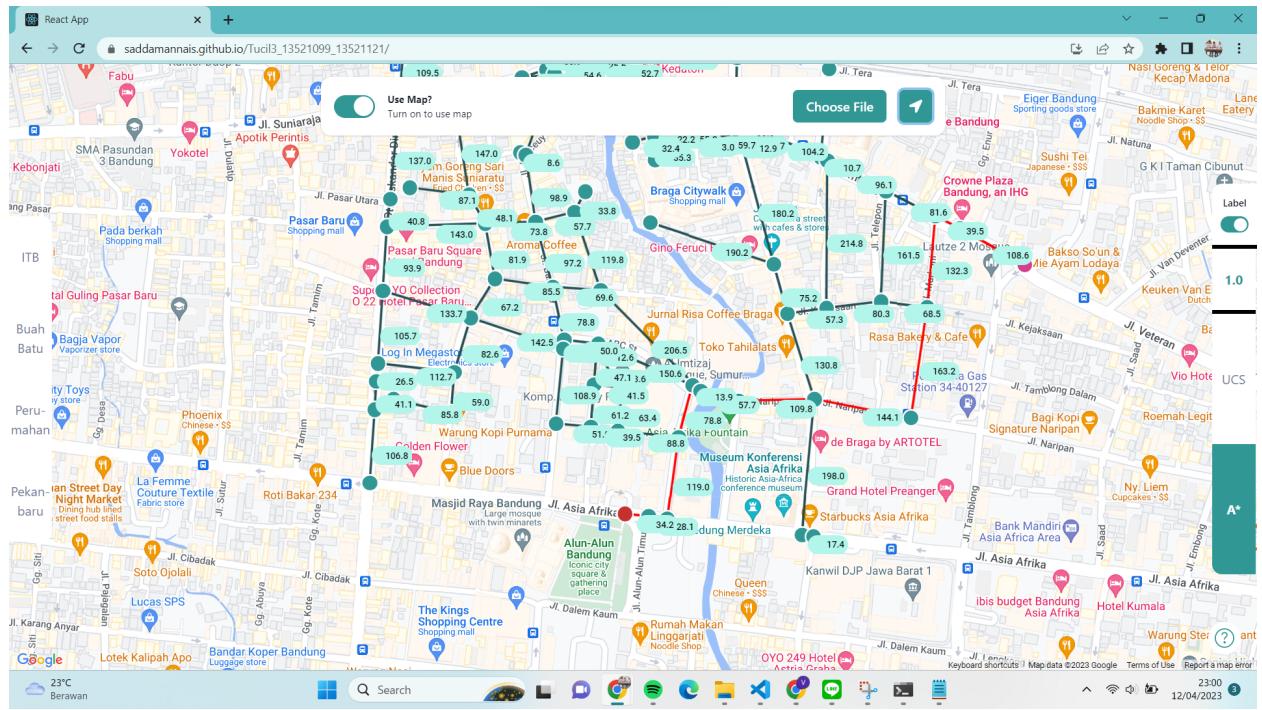


#### B. Spesifikasi Bonus

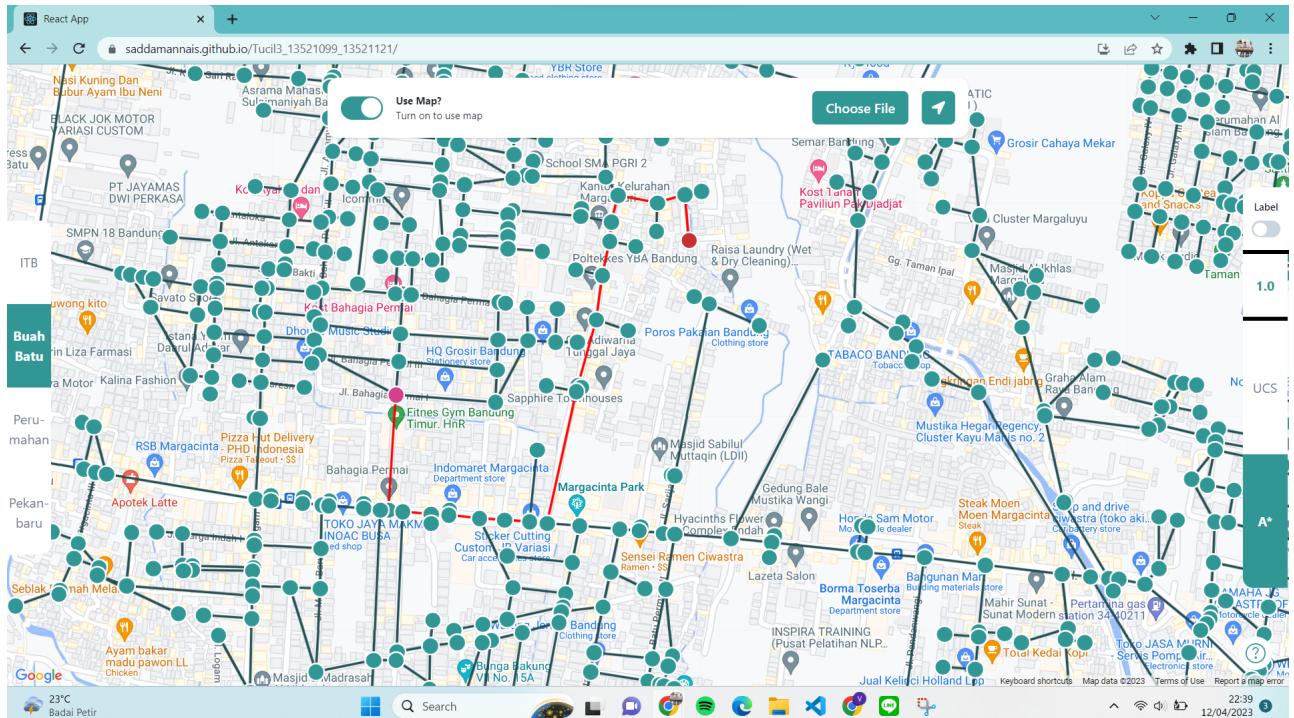
## 1. Perhitungan peta pada daerah ITB



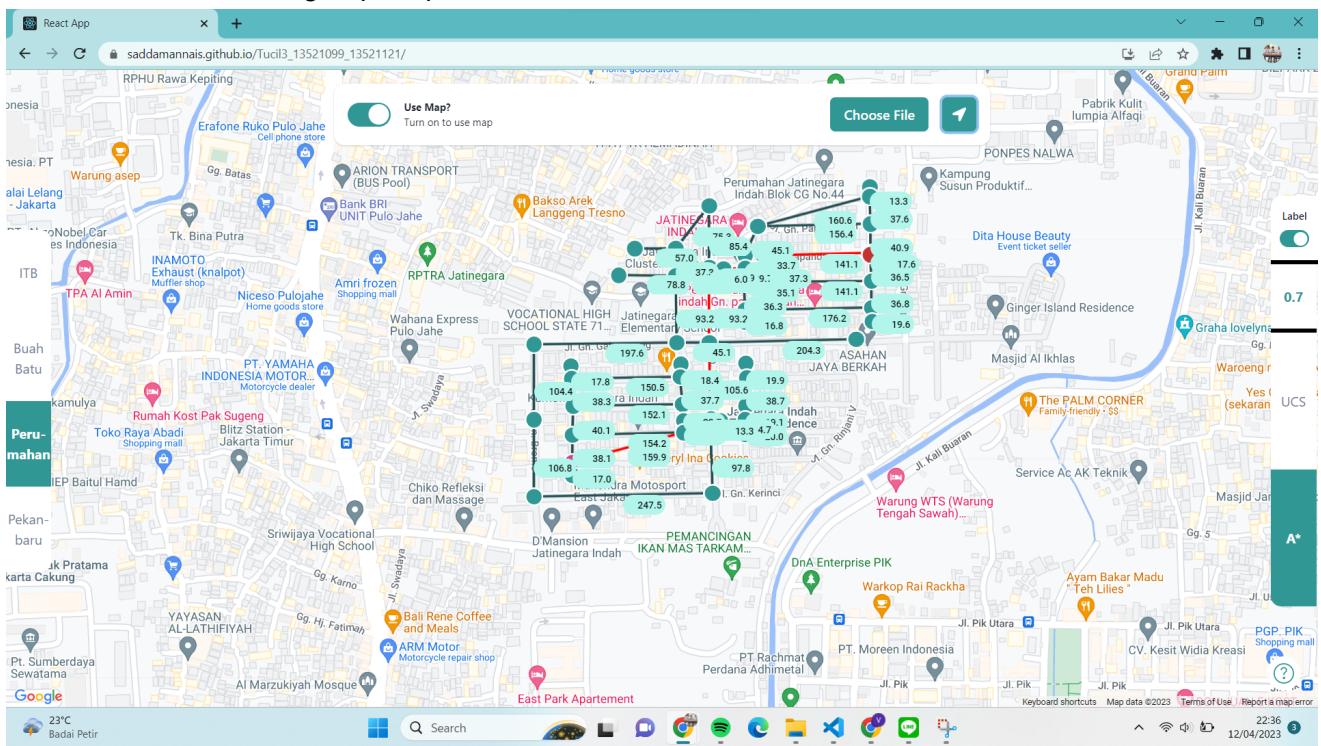
## 2. Perhitungan peta pada daerah Alun-alun Bandung



### 3. Perhitungan peta pada daerah Buahbatu



#### 4. Perhitungan peta pada kawasan asal



## TABLE

Poin	YA	TIDAK
1. Program dapat menerima input graf	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program dapat menghitung lintasan terpendek dengan UCS	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Program dapat menghitung lintasan terpendek dengan A*	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Program dapat menampilkan lintasan terpendek serta jaraknya	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## **Kesimpulan Saran dan Refleksi**

### **Kesimpulan**

Kami telah berhasil membuat sebuah aplikasi berbasis web yang dapat mencari rute terpendek dari satu titik ke titik lain menggunakan algoritma A\* dan Uniform Cost Search (UCS), sebagai bagian dari tugas kecil IF2211 Strategi Algoritma semester 2 2022/2023 dengan judul "Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek". Kami menggunakan bahasa pemrograman Javascript untuk merancang web dengan framework react dan menghitung jarak terpendeknya juga menggunakan javascript untuk format input file berupa txt. Untuk algoritma pencarian A\* bersifat heuristik, sedangkan UCS menggunakan metode uninformed search

### **Saran**

Pengembangan aplikasi web membutuhkan waktu yang cukup lama, terutama jika mencari API gratis untuk menampilkan peta. Karena itu, untuk masa depan, disarankan untuk merencanakan waktu yang cukup untuk pengembangan aplikasi web dan mencari referensi untuk mendapatkan API secara gratis. Selain itu, spesifikasi tugas sebaiknya diperjelas dan dihindari ambiguitas agar tidak menimbulkan kesalahpahaman.

### **Refleksi**

Refleksi dari kelompok kami mungkin terkait komunikasi dan sistem penggerjaan 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.

### **Tanggapan**

Tugas Kecil 3 IF2211 - Strategi Algoritma 2022/2023 terbukti menantang karena kami dituntut untuk membuat aplikasi berbasis web, mencari API peta, dan merancang algoritma A\* dan UCS dalam waktu yang terbatas. Perjuangan kami dalam menyelesaikan tugas tersebut sangat memacu kemampuan kami.

## **Lampiran**

Link Repository Github: [https://github.com/vierifirdaus/Tucil3\\_13521099\\_13521121](https://github.com/vierifirdaus/Tucil3_13521099_13521121)

Link implementasi dalam web :

[https://saddamannais.github.io/Tucil3\\_13521099\\_13521121/](https://saddamannais.github.io/Tucil3_13521099_13521121/)