

## **LAPORAN TUGAS KECIL 3 IF2211**

### **Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**

Ditujukan untuk memenuhi tugas kecil 3 mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023

Dibuat oleh :

Rizky Abdillah Rasyid      13521109  
Muh. Abdul Aziz Ghazali    13521128



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

## **Daftar Isi**

<b>Bab I Latar Belakang.....</b>	<b>3</b>
1.1 Algoritma Uniform Cost Search.....	3
1.2 Algoritma A star.....	3
1.3 Go Language.....	4
1.4 Typescript dan Angular.....	4
<b>Bab II Kode Program.....</b>	<b>5</b>
<b>Bab III Testing Program.....</b>	<b>22</b>
<b>Bab IV Kesimpulan dan Saran.....</b>	<b>29</b>
<b>Referensi.....</b>	<b>30</b>

## Bab I Latar Belakang

Algoritma UCS (Uniform cost search) dan A\* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Program yang kami buat dapat menentukan lintasan terpendek berdasarkan peta OpenLayers 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) dengan metode yang disediakan oleh API OpenLayers. Program kami menggunakan Go Language untuk *backend* dan Typescript dengan *framework* Angular untuk *frontend*.

### 1.1 Algoritma Uniform Cost Search

Algoritma UCS (Uniform Cost Search) adalah salah satu algoritma pencarian graf yang digunakan untuk mencari jalur dengan biaya terkecil atau jarak terpendek dari satu simpul ke simpul lainnya. Algoritma UCS sangat berguna untuk menyelesaikan permasalahan optimisasi yang melibatkan graf.

Pada dasarnya, algoritma UCS bekerja dengan cara mengunjungi simpul terdekat yang memiliki biaya atau jarak terkecil dari simpul awal. Kemudian, algoritma UCS akan mengembangkan simpul tersebut dengan mencari semua simpul yang dapat dicapai dari simpul tersebut, dan memperbarui biaya dan jarak ke simpul-simpul tersebut jika dibutuhkan. Algoritma UCS akan terus melakukan hal ini sampai mencapai simpul tujuan atau tidak ada simpul yang dapat dikunjungi lagi.

### 1.2 Algoritma A star

Algoritma A\* (A-star) adalah salah satu algoritma pencarian jalur terbaik (best-first search) untuk menemukan jalur terpendek atau terbaik antara dua titik pada graf, dengan mempertimbangkan biaya atau jarak dari simpul ke simpul.

Algoritma A\* memadukan pencarian graf berbasis biaya dan pencarian heuristik. Pencarian berbasis biaya digunakan untuk menghitung biaya yang dibutuhkan untuk mencapai suatu simpul dari simpul asal, sedangkan pencarian heuristik digunakan untuk memperkirakan biaya yang dibutuhkan untuk mencapai simpul tujuan dari simpul saat ini. Heuristik ini biasanya dinyatakan dalam bentuk fungsi estimasi biaya ( $h$ ) yang memperkirakan jarak sisa dari simpul saat ini ke simpul tujuan.

Algoritma A\* bekerja dengan menjelajahi graf dan mengembangkan simpul dengan biaya terkecil atau jarak terpendek dari simpul asal ke simpul tujuan. Selain itu, algoritma juga mempertimbangkan estimasi biaya sisa ( $h$ ) dari simpul saat ini ke simpul tujuan dalam menentukan urutan pengembangan simpul. Algoritma A\* kemudian memilih simpul dengan nilai  $f$  terkecil ( $f = g + h$ ), di mana  $g$  adalah biaya aktual untuk mencapai simpul saat ini dari simpul asal.

Salah satu keuntungan utama dari algoritma A\* adalah efisiensi dan kemampuannya untuk menemukan jalur terpendek atau terbaik dalam graf yang besar dan kompleks. Algoritma ini juga dapat diterapkan pada berbagai jenis masalah, seperti permasalahan perutean, permasalahan pencocokan pola, dan sebagainya.

Namun, untuk memastikan kinerja optimal dari algoritma A\*, diperlukan pemilihan heuristik yang tepat, yang dapat memperkirakan biaya sisa dengan cukup akurat tanpa terlalu memakan waktu atau sumber daya yang berlebihan. Oleh karena itu, pemilihan heuristik yang tepat sangat penting untuk memastikan keberhasilan algoritma A\* dalam menyelesaikan masalah yang diberikan.

### 1.3 Go Language

Go Language, juga dikenal sebagai Golang, adalah bahasa pemrograman open source yang dikembangkan oleh Google pada tahun 2007. Go Language merupakan bahasa pemrograman yang dapat digunakan untuk membangun berbagai jenis perangkat lunak, seperti aplikasi desktop, web, dan mobile. Bahasa ini juga memiliki keunggulan dalam hal kinerja, efisiensi, dan pengelolaan memori, sehingga cocok untuk digunakan dalam lingkungan perangkat keras yang terbatas.

Salah satu fitur unggulan dari Go Language adalah kemampuannya dalam pemrograman konkuren dan paralel. Bahasa ini menyediakan fitur konkurensi bawaan yang memungkinkan pengembang untuk membangun aplikasi yang dapat menjalankan beberapa tugas secara bersamaan, tanpa perlu mengalami masalah seperti deadlock atau race condition.

### 1.4 Typescript dan Angular

Angular adalah kerangka kerja (framework) open source untuk pengembangan aplikasi web yang dibuat dan dipelihara oleh Google. Angular memungkinkan pengembang untuk membangun aplikasi web yang dinamis dan responsif dengan menggunakan HTML, CSS, dan JavaScript.

Salah satu keunggulan Angular adalah kemampuannya dalam memanipulasi dokumen HTML secara dinamis dan membuat aplikasi web yang interaktif dengan mudah. Angular juga menyediakan fitur-fitur seperti data binding, modularitas, routing, dan pengujian otomatis yang mempermudah proses pengembangan dan perawatan aplikasi web.

## Bab II Kode Program

Node.go

```
1 package DataType
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 type Node struct {
9     Id   int
10    Name string
11    Coor struct {
12        X float64
13        Y float64
14    }
15 }
16
17 func (n Node) Distance(o Node) float64 {
18     return math.Sqrt(math.Pow(n.Coor.X-o.Coor.X, 2) + math.Pow(n.Coor.Y-o.Coor.Y, 2))
19 }
20
21 func (n Node) DistanceHaversine(o Node) float64 {
22     lat1 := n.Coor.Y
23     lat2 := o.Coor.Y
24     lon1 := n.Coor.X
25     lon2 := o.Coor.X
26
27     rad := func(x float64) float64 {
28         return x * math.Pi / 180
29     }
30
31     var R float64 = 6371000 // km
32     dLat := rad(lat2 - lat1)
33     dLon := rad(lon2 - lon1)
34     lat1 = rad(lat1)
35     lat2 = rad(lat2)
36
37     a := math.Sin(dLat/2)*math.Sin(dLat/2) + math.Sin(dLon/2)*math.Sin(dLon/2)*math.Cos(lat1)*math.Cos(lat2)
38     c := 2 * math.Atan2(math.Sqrt(a), math.Sqrt(1-a))
39     d := R * c
40
41     return d
42 }
43
44 func (n Node) PrintNode() {
45     fmt.Printf("ID : %d\n", n.Id)
46     fmt.Printf("Nama : %s\n", n.Name)
47     fmt.Printf("Coor : (%f, %f)\n", n.Coor.X, n.Coor.Y)
48 }
49
```

## Graph.go

```
1 package DataType
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 type Link struct {
9     To    *Node
10    Dist float64
11 }
12
13 type Graph struct {
14     Nodes []*Node
15     Links map[*Node][]*Link
16 }
17
18 // READ FILE
19 func Json2Nodes(jsonData []byte, nodes *[]Node) ([][]int, int, int, string) {
20     var parse struct {
21         Nodes []*struct {
22             ID   int      `json:"id"`
23             Name string   `json:"name"`
24             Coor [2]float64 `json:"coor"`
25         } `json:"nodes"`
26         AdjMat [][]int `json:"mat"`
27         From  int      `json:"from"`
28         To    int      `json:"to"`
29         Algo  string   `json:"algo"`
30     }
31
32     err := json.Unmarshal(jsonData, &parse)
33     if err != nil {
34         panic(err)
35     }
36
37     *nodes = make([]*Node, len(parse.Nodes))
38     for i := 0; i < len(*nodes); i++ {
39         (*nodes)[i] = Node{Id: parse.Nodes[i].ID, Name: parse.Nodes[i].Name, Coor: struct {
40             X float64
41             Y float64
42         }{Y: parse.Nodes[i].Coor[0], X: parse.Nodes[i].Coor[1]}}
43     }
44     return parse.AdjMat, parse.From, parse.To, parse.Algo
45 }
46
47 func (g *Graph) CreateGraph(nodes []Node, mat [][]int) {
48     g.Nodes = make([]*Node, len(nodes))
49     for i := 0; i < len(nodes); i++ {
50         g.Nodes[i] = &nodes[i]
51     }
52
53     g.Links = make(map[*Node][]*Link)
54
55     for i := 0; i < len(mat); i++ {
56         for j := 0; j < len(mat[0]); j++ {
57             if mat[i][j] == 1 {
58                 g.Links[g.Nodes[i]] = append(g.Links[g.Nodes[i]], &Link{To: g.Nodes[j], Dist: nodes[i].DistanceHaversine(nodes[j])})
59             }
60         }
61     }
62 }
63
64 func (l Link) PrintLink() {
65     l.To.PrintNode()
66     fmt.Printf("Dist : %f\n", l.Dist)
67 }
68
69 func (g Graph) PrintGraph() {
70     for i := 0; i < len(g.Nodes); i++ {
71         g.Nodes[i].PrintNode()
72     }
73
74     for key, link := range g.Links {
75         fmt.Println("Parent : ")
76         key.PrintNode()
77         for _, l := range link {
78             l.PrintLink()
79         }
80     }
81 }
```

## PrioQueue.go

```
● ● ●

1 package DataType
2
3 type Route struct {
4     Cost float64
5     Nodes []*Node
6 }
7
8 type PrioQueue []*Route
9
10 func (pq PrioQueue) Less(i, j int) bool {
11     return pq[i].Cost < pq[j].Cost
12 }
13
14 func (pq *PrioQueue) Swap(i, j int) {
15     temp := (*pq)[j]
16     (*pq)[j] = (*pq)[i]
17     (*pq)[i] = temp
18 }
19
20 func (pq *PrioQueue) Push(newRoute Route) {
21     *pq = append(*pq, &newRoute)
22     canSwap := true
23     for i := len(*pq) - 1; i > 0 && canSwap; i-- {
24         if pq.Less(i, i-1) {
25             pq.Swap(i, i-1)
26         } else {
27             canSwap = false
28         }
29     }
30 }
31
32 func (pq *PrioQueue) Pop() Route {
33     Top := Route{Cost: (*pq)[0].Cost, Nodes: (*pq)[0].Nodes}
34     *pq = (*pq)[1:]
35     return Top
36 }
37
```

## Algorithm.go

```
1 package DataType
2
3 func UCS(g *Graph, From, To *Node) ([]*Node, float64) {
4     queue := PrioQueue{&Route{Nodes: []*Node{From}}}
5     visit := make(map[*Node]bool)
6     for len(queue) > 0 {
7         currRoute := queue.Pop()
8         currNode := currRoute.Nodes[len(currRoute.Nodes)-1]
9         if currNode == To {
10             return currRoute.Nodes, currRoute.Cost
11         }
12         // Bypass if current node has been visited before to avoid loop
13         if visit[currNode] {
14             continue
15         }
16         visit[currNode] = true
17         for _, link := range g.Links[currNode] {
18             newRoute := &Route{Cost: currRoute.Cost + link.Dist}
19             newRoute.Nodes = make([]*Node, len(currRoute.Nodes))
20             copy(newRoute.Nodes, currRoute.Nodes)
21             newRoute.Nodes = append(newRoute.Nodes, link.To)
22             queue.Push(*newRoute)
23         }
24     }
25     return nil, -1
26 }
27
28 func Astar(g *Graph, From, To *Node, nodes []Node) ([]*Node, float64) {
29     heur := CreateHeuristic(nodes, To)
30     queue := PrioQueue{&Route{Nodes: []*Node{From}, Cost: heur[From]}}
31     visited := make(map[*Node]bool)
32
33     for len(queue) > 0 {
34         currRoute := queue.Pop()
35         currNode := currRoute.Nodes[len(currRoute.Nodes)-1]
36
37         // Check if current node has been visited before
38         if visited[currNode] {
39             continue
40         }
41
42         visited[currNode] = true
43
44         if currNode == To {
45             return currRoute.Nodes, currRoute.Cost
46         }
47         for _, link := range g.Links[currNode] {
48             newRoute := &Route{Cost: currRoute.Cost + link.Dist}
49             newRoute.Nodes = make([]*Node, len(currRoute.Nodes))
50             copy(newRoute.Nodes, currRoute.Nodes)
51             newRoute.Nodes = append(newRoute.Nodes, link.To)
52             newRoute.Cost += heur[link.To]
53             queue.Push(*newRoute)
54         }
55     }
56     return nil, -1
57 }
58
59 func CreateHeuristic(nodes []Node, To *Node) map[*Node]float64 {
60     heurist := make(map[*Node]float64)
61     for _, node := range nodes {
62         // Using haversine calculation to increase accuracy
63         heurist[&node] = node.DistanceHaversine(*To)
64     }
65     return heurist
66 }
```

```
● ● ●  
1 package DataType  
2  
3 type Message struct {  
4     Route    string  
5     Distance float64  
6     Nodes    []*Node  
7     Status   bool  
8 }  
9  
10 func (m *Message) SetDefault() {  
11     m.Route = ""  
12     m.Distance = 0  
13     m.Nodes = nil  
14     m.Status = false  
15 }  
16
```

## main.go

```
1 package main
2
3 import (
4     Control "Backend/Controller"
5     DataType "Backend/Model"
6     "github.com/gin-contrib/cors"
7 )
8 import "github.com/gin-gonic/gin"
9
10 func main() {
11     r := gin.Default()
12
13     r.Use(cors.Default())
14
15     var data DataType.Graph
16     var msg DataType.Message
17
18     r.POST("/", Control.PostHandler(&data, &msg))
19     r.GET("/", Control.GetHandler(&data, &msg))
20
21     r.Run()
22 }
23
```

## main.component.html

```
<a routerLink="/"></a>

<link
  rel="stylesheet"
  href="https://cdn.rawgit.com/openlayers/openlayers.github.io/master/en/v5.3.0/css/ol.css"
  type="text/css"
/>

<div id="sidebar">
  <div class="container">
    <h2 class="text-center mb-4">INPUT</h2>
    <div class="mb-1">
      <label class=""><h5>Input File</h5></label>
      <input type="file" accept=".json"
        (change)="onFileSelected($event)" />
      <div class="form-text">Hanya menerima input file bertipe
        .json</div>
    </div>
    <div class="mb-1">
      <label class=""><h5>ID Titik Asal</h5></label>
      <input #Asal type="text" />
    </div>
    <div class="mb-3">
      <label class=""><h5>ID Titik Tujuan</h5></label>
      <input #Tujuan type="text" />
      <div class="form-text">
        ID harus tepat, jika ID tidak ditemukan pada file JSON maka
        program akan
        Error
      </div>
    </div>
    <div class="mb-1">
      <label class=""><h5>Pilihan Algoritma</h5></label>
      <div class="mb-0">
        <label>
          <input
            type="radio"
            name="Algorithm"
            value="UCS">
        </label>
      </div>
    </div>
  </div>
</div>
```

```

        (change)="onAlgoChanged($event)"
    />
    UCS
</label>
</div>
<div class="mb-3">
    <label>
        <input
            type="radio"
            name="Algorithm"
            value="A*"
            (change)="onAlgoChanged($event)"
        />
        A*
    </label>
</div>
</div>
<!-- <button class="btn btn-primary" (click)="-->
<button class="btn btn-primary" (click)="onEnter(Asal.value,
Tujuan.value)">
    Submit
</button>
<div class="form-text mb-3">
    Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan
    gambar graf
    pada peta
</div>
<div class="mb-1">
    <button class="btn btn-primary"
(click)="showRoute()">Route</button>
</div>
<div class="form-text">
    Klik ketika gambar graf sudah terlihat, untuk melihat gambaran
    rute
    perjalanan anda
</div>
</div>
</div>

<!-- Mainbar -->
<div id="mainbar">
    <div class="container">

```

```
<h2 class="mb-3">
    Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan
Terpendek
</h2>
<div class="solutionContainer">
    <div id="map" class="map"></div>
</div>
<div class="result">
    <div class="route mt-4">
        <h5>Rute : {{ RouteView }}</h5>
    </div>
    <div class="dist mt-5">
        <h5>Jarak Tempuh : {{ RouteDistance }} meter</h5>
    </div>
</div>
</div>
```

### main.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import Map from 'ol/Map';
import View from 'ol/View';
import TileLayer from 'ol/layer/Tile';
import OSM from 'ol/source/OSM';
import { fromLonLat } from 'ol/proj';

import Feature from 'ol/Feature';
import Point from 'ol/geom/Point';
import LineString from 'ol/geom/LineString';
import { Vector } from 'ol/source';
import { Vector as VectorLayer } from 'ol/layer';
import { Style, Stroke, Circle, Fill, Text } from 'ol/style';

@Component({
  selector: 'app-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css'],
})
export class MainComponent implements OnInit {
  MessageJSON: any = null;
  RouteView: string = 'NaN';
  RouteDistance: string = 'NaN';
  Algorithm: string = '';
  Solution: any = null;

  map: Map = new Map({ });
  markerLayer = new VectorLayer({ });
  lineLayer = new VectorLayer({ });

  constructor(private http: HttpClient) {}

  ngOnInit(): void {
    this.initMap();
    this.initLineLayer();
    this.initMarkerLayer();
  }
}
```

```

private initMap(): void {
    this.map = new Map({
        target: 'map',
        layers: [
            new TileLayer({
                source: new OSM(),
            }),
        ],
        view: new View({
            center: fromLonLat([0, 0]),
            zoom: 0,
        }),
    });
}

draw2MarkersAndLine(
    coor1: any,
    name1: string,
    coor2: any,
    name2: string,
    isSolution: boolean
) {
    var marker1: Feature = new Feature({});
    var marker2: Feature = new Feature({});
    const markerSource = this.markerLayer.getSource();

    // add first marker
    marker1 = new Feature({
        geometry: new Point(fromLonLat([coor1[1], coor1[0]])),
        name: name1,
    });
    // add second marker
    marker2 = new Feature({
        geometry: new Point(fromLonLat([coor2[1], coor2[0]])),
        name: name2,
    });

    markerSource?.addFeature(marker1);
    markerSource?.addFeature(marker2);

    // add line between markers
    const lineSource = this.lineLayer.getSource();
}

```

```

var color = 'blue';
if (isSolution == true) {
    color = 'green';
}

if (lineSource && marker1.getGeometry() && marker2.getGeometry())
{
    const line = new Feature({
        geometry: new LineString([
            (<Point>marker1.getGeometry()).getCoordinates(),
            (<Point>marker2.getGeometry()).getCoordinates(),
        ]),
        nama: calculateDistance(coor1[1], coor1[0], coor2[1],
coor2[0]),
        warna: color,
    });
    lineSource.addFeature(line);
}
}

onAlgoChanged(event: Event) {
    this.Algorithm = (event.target as HTMLInputElement).value;
}
onFileSelected(event: any) {
    const file = event.target.files[0];
    const reader = new FileReader();
    reader.readAsText(file);
    reader.onload = () => {
        const fileContent = reader.result;
        this.MessageJSON = JSON.parse(fileContent as string);
    };
}

private initMarkerLayer(): void {
    const markerSource = new Vector({
        features: [],
    });
    const markerStyle = new Style({
        image: new Circle({
            radius: 7,

```

```

        fill: new Fill({ color: 'red' }),
        stroke: new Stroke({
            color: 'white',
            width: 2,
        }) ,
    ) ,
    text: new Text({
        text: '',
        font: '12px sans-serif',
        fill: new Fill({ color: 'black' }),
        stroke: new Stroke({
            color: 'white',
            width: 2,
        }) ,
        offsetY: -15, // mengatur posisi teks ke atas
    ) ,
);

this.markerLayer = new VectorLayer({
    source: markerSource,
    style: function (feature) {
        markerStyle.getText().setText(feature.get('nama'));
        return markerStyle;
    },
});

this.map.addLayer(this.markerLayer);
}

private initLineLayer(): void {
    const lineSource = new Vector({
        features: [],
    });

    const lineStyle = new Style({
        stroke: new Stroke({
            color: 'blue',
            width: 5,
        }) ,
        text: new Text({
            text: '',
            font: '12px sans-serif',
        })
    });
}

```

```

        fill: new Fill({ color: 'black' }),
        stroke: new Stroke({
            color: 'white',
            width: 2,
        }),
        offsetY: -15, // mengatur posisi teks ke atas
    ) ,
);
}

this.lineLayer = new VectorLayer({
    source: lineSource,
    style: function (feature) {
        lineStyle.getText().setText(feature.get('nama'));
        lineStyle.getStroke().setColor(feature.get('warna'));
        return lineStyle;
    },
});
this.map.addLayer(this.lineLayer);
}

createGraph(matrix: any, node: any, size: number) {
    for (let i: number = 0; i < size; i++) {
        for (let j: number = 0; j < size; j++) {
            if (matrix[i][j] == 1) {
                this.draw2MarkersAndLine(
                    node[i].coor,
                    node[i].name,
                    node[j].coor,
                    node[j].name,
                    false
                );
            }
        }
    }
}

createSolution() {
    for (let i = 1; i < this.Solution.result.length; i++) {
        console.log(this.Solution.result);

        var a = [this.Solution.result[i].Coor.Y,

```

```

        this.Solution.result[i].Coor.X];
        var b = [
            this.Solution.result[i - 1].Coor.Y,
            this.Solution.result[i - 1].Coor.X,
        ];

        this.draw2MarkersAndLine(
            a,
            this.Solution.result[i].Name,
            b,
            this.Solution.result[i - 1].Name,
            true
        );
    }
}

async showRoute() {
    console.log(this.Solution);
    await this.createSolution();
}

async onEnter(idStart: string, idDest: string) {
    var dest = parseInt(idDest);
    var strt = parseInt(idStart);

    try {
        if (this.MessageJSON.nodes.length < 8) {
            throw new Error('Jumlah Node kurang dari 8');
        }
        if (
            strt >= this.MessageJSON.nodes.length ||
            dest >= this.MessageJSON.nodes.length
        ) {
            throw new Error(
                'Input Titik Asal atau Tujuan Tidak Ditemukan, Cek Input
File atau Titik kembali'
            );
        }
        if (strt < 0 || dest < 0) {
            throw new Error('Tidak Ada Id Negatif');
        }
        if (this.Algorithm == '') {

```

```

        throw new Error('Algoritma belum dipilih');
    }
} catch (error: any) {
    alert(error.message);
    setTimeout(() => {
        location.reload();
    }, 1000); // menunggu 5 detik sebelum merefresh halaman
    return;
}
// const size: number = this.MessageJSON.node.length;
await this.createGraph(
    this.MessageJSON.mat,
    this.MessageJSON.nodes,
    this.MessageJSON.nodes.length
);

this.MessageJSON.from = strt;
this.MessageJSON.to = dest;
this.MessageJSON.algo = this.Algorithm;

await this.http
    .post('http://localhost:8080/', this.MessageJSON)
    .toPromise();

await this.http
    .get('http://localhost:8080/', { responseType: 'json' })
    .subscribe((data: any) => {
        console.log(data);
        this.RouteDistance = '';
        this.RouteView = '';
        this.Solution = data.result;
        this.RouteDistance = data.distance;
        this.RouteView = data.routestr;
        this.Solution = data;
    });
}

function calculateDistance(
    lat1: number,
    lon1: number,
    lat2: number,

```

```
lon2: number
) {
    const R = 6371e3; // radius of the earth in meters
    const phi1 = (lat1 * Math.PI) / 180;
    const phi2 = (lat2 * Math.PI) / 180;
    const deltaPhi = ((lat2 - lat1) * Math.PI) / 180;
    const deltaLambda = ((lon2 - lon1) * Math.PI) / 180;

    const a =
        Math.sin(deltaPhi / 2) * Math.sin(deltaPhi / 2) +
        Math.cos(phi1) *
            Math.cos(phi2) *
            Math.sin(deltaLambda / 2) *
            Math.sin(deltaLambda / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    const d = R * c;
    return d.toFixed(2) + ' meters';
}
```

Kode Program tidak dimasukan semua hanya yang inti saja, karena banyak kode yang merupakan bawaan Template Framework

### Bab III Testing Program

Graf yang diterima program ini memiliki format JSON yang berisi informasi seluruh *nodes* berupa id, nama, dan koordinat, kemudian informasi matriks ketetanggaan untuk merepresentasikan keterhubungan seluruh *nodes*. Berikut adalah contoh file JSON untuk *input* program :

```
{
  "nodes": [
    {
      "id": 0,
      "name": "A",
      "coor": [0,0]
    },
    {
      "id": 1,
      "name": "B",
      "coor": [2,2]
    },
    {
      "id": 2,
      "name": "C",
      "coor": [4,0]
    },
    {
      "id": 3,
      "name": "D",
      "coor": [1,3]
    },
    {
      "id": 4,
      "name": "E",
      "coor": [4,4]
    }
  ],
  "mat": [
    [1, 1, 0, 0, 0],
    [1, 1, 1, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 1],
    [1, 0, 1, 0, 1]
  ]
}
```

```

        [0, 0, 1, 1, 1],
        [0, 0, 0, 1, 1]
    ]
}

```

Berikut ini adalah *screenshot* dari program dan *testcase* yang digunakan :

### 1. Tampilan awal program

**INPUT**

**Input File**  
 No file chosen  
 Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**

ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : NaN

Jarak Tempuh : NaN meter

### 2. Daerah Alun-Alun dengan UCS

**INPUT**

**Input File**  
 Alun-Alun.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**

ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : Dalem Kaum - Dewi Sartika - Otto - Cibadak - Dalem Kaum - Norsefiicden

Jarak Tempuh : 399.8777858157339 meter

### 3. Daerah Alun-Alun dengan A\*

**INPUT**

**Input File**  
 Alun-Alun.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**  
  
**ID Titik Tujuan**

ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**

Rute : Dalem Kaum - Dewi Sartika - Otto - Cibadak - Dalem Kaum - Norseiiden

Jarak Tempuh : 399.8777858157339 meter

### 4. Daerah Buahbatu dengan UCS

**INPUT**

**Input File**  
 Buahbatu.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**  
  
**ID Titik Tujuan**

ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**

Rute : Ibrahim Adjie , Terusan Buah Batu - Tugu Kordon - Samudra Bangunan - Waroeng Abror - Klinik Bidan Apriani

Jarak Tempuh : 3898.9694692861185 meter

## 5. Buahbatu dengan A\*

**INPUT**

**Input File**  
 Buahbatu.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**

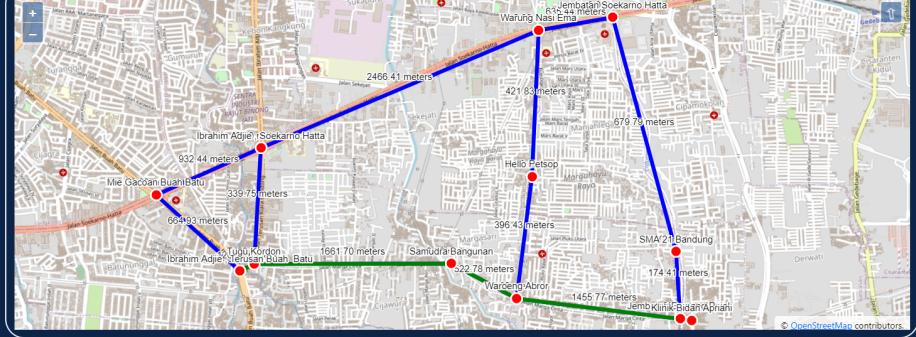
**ID Titik Tujuan**  
  
 ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**  
 Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**  
 Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : Ibrahim Adjie , Terusan Buah Batu - Tugu Kordon - Samudra Bangunan - Waroeng Abror - Klinik Bidan Apriani

Jarak Tempuh : 3898.9694692861185 meter

## 6. Sekitar ITB dengan UCS

**INPUT**

**Input File**  
 sekitarDago.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**

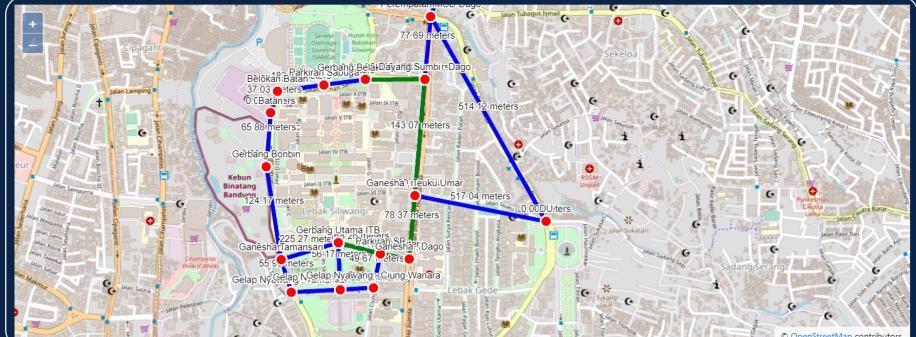
**ID Titik Tujuan**  
  
 ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**  
 Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**  
 Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : Gerbang Utama ITB - Parkiran SR - Ganesha , Dago - Ganesha , Teuku Umar - Dayang Sumbi , Dago - Gerbang Belakang ITB

Jarak Tempuh : 1223.7564424439297 meter

## 7. Sekitar ITB dengan A\*

**INPUT**

**Input File**  
 sekitarDago.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**  
  
 ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

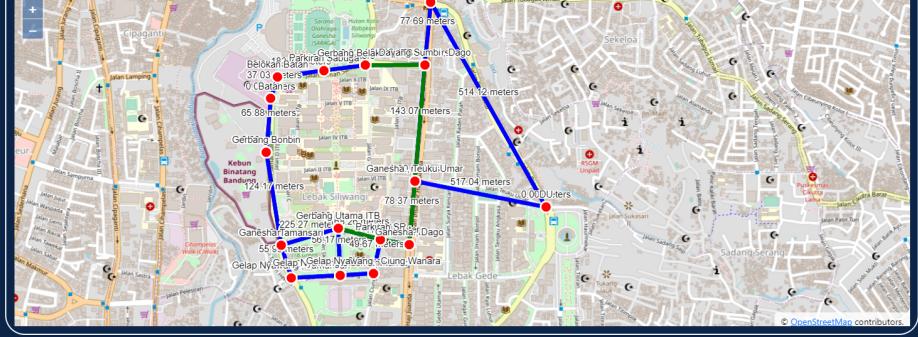
**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : Gerbang Utama ITB - Parkiran SR - Ganesha , Dago - Ganesha , Teuku Umar - Dayang Sumbi , Dago - Gerbang Belakang ITB

Jarak Tempuh : 1223.7564424439297 meter

## 8. Sekitar UGM dengan UCS

**INPUT**

**Input File**  
 UGMmaszeh.json  
 Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**  
  
 ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

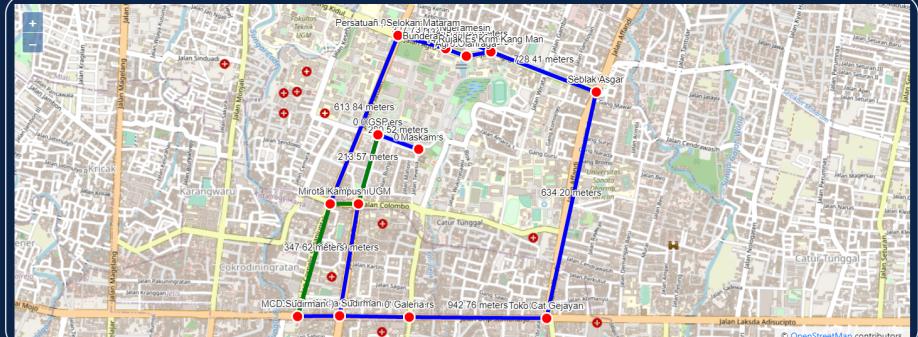
**Submit**

Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**

Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**



Rute : GSP - Bunderan UGM - Mirota Kampus - MCD Sudirman :)

Jarak Tempuh : 1477.8561771128675 meter

## 9. Sekitar UGM dengan A\*

**INPUT**

**Input File**  
 UGMmaszeh.json  
Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**  
  
ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**  
Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**  
Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

**Implementasi Algoritma UCS dan A\* untuk Menentukan Lintasan Terpendek**

Rute : GSP - Bunderan UGM - Gramedia Sudirman - Galeria

Jarak Tempuh : 1737.3100230260059 meter

## 10. Input JSON salah

**INPUT**

**Input File**  
 Input salah.json  
Hanya menerima input file bertipe json

**ID Titik Asal**

**ID Titik Tujuan**  
  
ID harus tepat, jika ID tidak ditemukan pada file JSON maka program akan Error

**Pilihan Algoritma**  
 UCS  
 A\*

**Submit**  
Jika inputan sudah lengkap, Submit untuk mendapatkan rute dan gambar graf pada peta

**Route**  
Klik ketika gambar graf sudah terlihat, untuk melihat gambaran rute

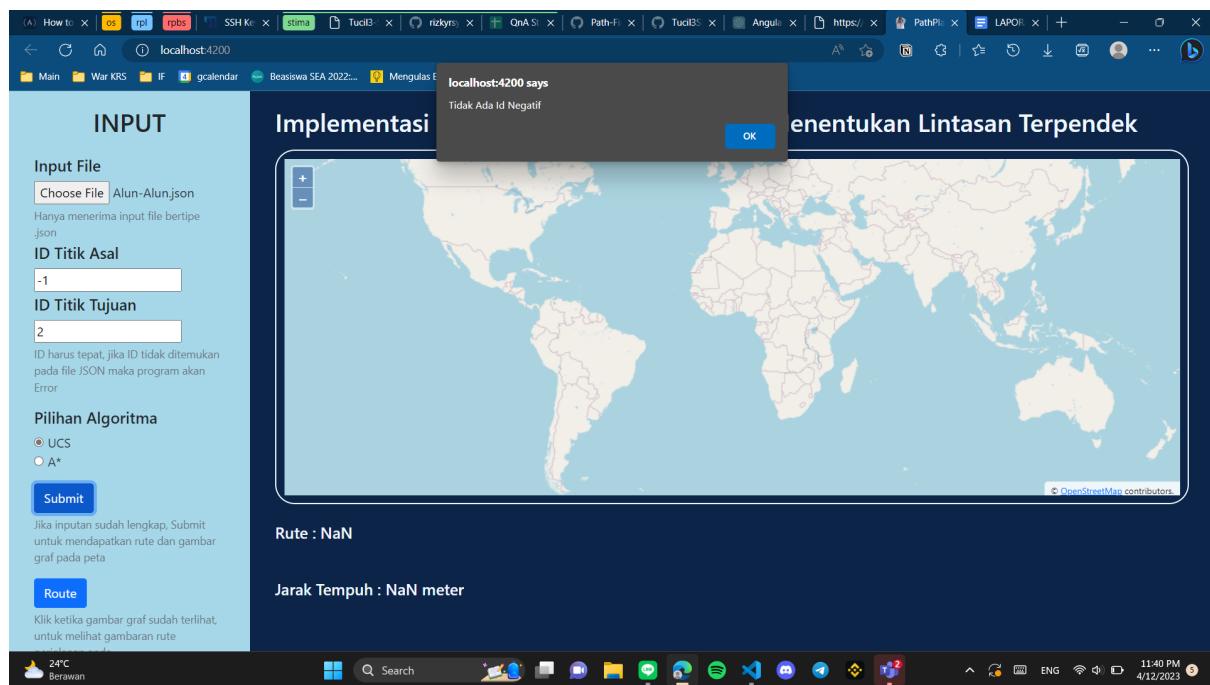
**localhost:4200 says**  
Jumlah Node kurang dari 8

**Implementasi**

Rute : GSP - Bunderan UGM - Gramedia Sudirman - Galeria

Jarak Tempuh : 1737.3100230260059 meter

## 11. ID invalid



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	✓

## Bab IV Kesimpulan dan Saran

Program yang kami buat telah berhasil mengimplementasikan algoritma A star dan Uniform Cost Search untuk mencari rute terpendek dengan akurat dan cepat dengan *framework* Angular dan bahasa Go. *Error* yang kemungkinan terjadi sudah dihandle oleh program sehingga program tidak akan *crash*. Tampilan UI program sangat mudah untuk dimengerti oleh pengguna dan sudah *responsive*.

Berdasarkan pengujian yang telah dilakukan, dapat disimpulkan bahwa penggunaan algoritma A\* dan UCS memberikan hasil yang berbeda dalam mencari rute terbaik di peta OpenLayers. A\* cenderung lebih cepat dan lebih efisien dalam menemukan rute terbaik karena menggunakan heuristik yang memandu pencarian menuju tujuan, sementara UCS melakukan eksplorasi secara menyeluruh pada seluruh simpul yang ada dalam graf, meskipun pada akhirnya juga dapat menemukan rute terbaik.

Namun, penggunaan A\* dapat memakan waktu lebih lama jika heuristik yang digunakan tidak akurat atau tidak memandu pencarian dengan baik. Oleh karena itu, sangat penting untuk memilih heuristik yang sesuai untuk aplikasi yang sedang dibuat. Selain itu, penggunaan algoritma A\* juga dapat lebih sulit dalam implementasinya karena memerlukan perhitungan heuristik dan penanganan kondisi khusus.

Dalam hal ini, saran yang dapat diberikan adalah untuk mempertimbangkan baik-baik penggunaan algoritma A\* dan UCS sesuai dengan kondisi dan tujuan penggunaan aplikasi. Jika kecepatan dan efisiensi adalah faktor yang penting, maka A\* mungkin lebih cocok. Namun, jika keakuratan dan eksplorasi seluruh kemungkinan rute adalah hal yang lebih penting, maka UCS mungkin menjadi pilihan yang lebih tepat. Selain itu, memilih heuristik yang sesuai dan memperhatikan kasus khusus dalam implementasi A\* juga penting untuk memastikan keberhasilan pencarian rute terbaik.

## Referensi

<https://angular.io/>

<https://gin-gonic.com/>

<https://openlayers.org/>

Penentuan rute (Route/Path Planning) - Bagian 1. Diakses pada 12 April 2023, dari  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm>

Penentuan rute (Route/Path Planning) - Bagian 2. Diakses pada 12 April 2023, dari  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm>

Pranala Github : [rizkrysdy28/Tucil3-Path-Planning: Tugas Kecil 3 IF2211 Strategi Algoritma Semester II Tahun 2022/2023 Implementasi Algoritma UCS dan A\\* untuk Menentukan Lintasan Terpendek \(github.com\)](https://github.com/rizkrysdy28/Tucil3-Path-Planning)