

IF2211 Strategi Algoritma
IMPLEMENTASI ALGORITMA PENCARIAN UCS DAN A*
UNTUK PENCARIAN RUTE TERDEKAT DALAM PETA
Laporan Tugas Kecil III
Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester II (dua) Tahun Akademik 2022/2023



Oleh
Kenneth Ezekiel Suprantoni 13521089
Michael Jonathan Halim 13521124

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023

DAFTAR ISI

DAFTAR ISI	2
BAB I DESKRIPSI MASALAH	3
BAB II TEORI SINGKAT	4
2.1. Algoritma Uniform Cost Search	4
2.2. Algoritma A*	5
BAB III RANCANGAN DAN IMPLEMENTASI PROGRAM	6
3.1 Rancangan Algoritma	6
3.2 Rancangan Aplikasi	16
BAB IV HASIL PENGUJIAN	22
BAB V KESIMPULAN DAN SARAN	26
DAFTAR PUSTAKA	27
LAMPIRAN	28

BAB I

DESKRIPSI MASALAH

Pada tugas ini, permasalahan yang akan dibahas merupakan pencarian rute terdekat diberikan sebuah graf berbobot tak berarah atau peta. Pencarian rute terdekat ini akan diselesaikan menggunakan algoritma *Uniform Cost Search* dan *A**. Penyelesaian dari permasalahan ini akan diselesaikan dengan sebuah aplikasi berbasis *web*. Adapun dalam tugas ini juga akan digunakan sebuah API yang akan menunjukkan sebuah peta yang akan digunakan sebagai dasar dari pencarian rute.

Berikut merupakan spesifikasi lengkap dari deskripsi permasalahan yang diberikan: 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 ketetanggan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

BAB II

TEORI SINGKAT

2.1. Algoritma *Uniform Cost Search*

Algoritma *Uniform Cost Search* (UCS) merupakan algoritma pencarian rute secara *uninformed*, yang berarti tidak ada informasi yang tersedia terkait hubungan *state* sekarang dengan *goal state*. Algoritma UCS dapat digunakan sebagai alternatif dari BFS dan IDS jika jumlah langkah tidak mencerminkan biaya langkah, atau saat graf mempunyai bobot. Di dalam algoritma UCS, dapat digunakan sebuah *priority queue* dengan *priority* dari sebuah simpul adalah:

$$g(n) = \text{total bobot dari akar ke simpul } n$$

Alur utama algoritma UCS adalah sebagai berikut:

```
Function UCS() -> List of Simpul
{Algoritma UCS, mengembalikan sebuah list of simpul rute}
Deklarasi
PriorityQueue PQ
boolean found
Simpul currentNode, tetangga
List of Simpul Rute
Integer gn

Algoritma
// g(n) adalah jarak currentNode dari akar tambah jarak currentNode
ke tetangga

while (PQ is not empty and goal not found) do
    currentNode = PQ.dequeue
    if currentNode == goalNode then
        found = true
        break

    foreach tetangga : currentNode.tetangga
        gn = currentNode.cost + currentNode.distanceTo(tetangga)
        if tetangga is visited and tetangga.lowest_cost < gn then skip
        else
            Update tetangga.lowest_cost = gn
            Rute = currentNode.rute + tetangga
            PQ.enqueue(gn, tetangga, Rute)

    if found then return Rute
else return []
```

2.2. Algoritma A*

Algoritma A^* merupakan algoritma pencarian rute secara *informed*, yang berarti tersedia informasi terkait hubungan *state* sekarang dengan *goal state*. Dalam kasus pencarian rute, Algoritma A^* mengestimasikan keterkaitan hubungan *state* dengan *goal state* dengan estimasi jarak garis lurus dari simpul sekarang ke simpul tujuan. Di dalam algoritma A^* , dapat digunakan sebuah *priority queue* dengan *priority* dari sebuah simpul adalah:

$$\begin{aligned}g(n) &= \text{total bobot dari akar ke simpul } n \\h(n) &= \text{jarak simpul } n \text{ ke simpul tujuan} \\f(n) &= g(n) + h(n): f(n) \text{ adalah prioritas simpul } n\end{aligned}$$

Alur utama algoritma A^* adalah sebagai berikut:

```
Function AStar() -> List of Simpul
{Algoritma A*, mengembalikan sebuah list of simpul rute}
Deklarasi
PriorityQueue PQ
boolean found
Simpul currentNode, tetangga
List of Simpul Rute
Integer gn, hn

Algoritma
// g(n) adalah jarak currentNode dari akar tambah jarak currentNode
ke tetangga
// h(n) adalah jarak tetangga ke goalNode

while (PQ is not empty and goal not found) do
    currentNode = PQ.dequeue
    if currentNode == goalNode then
        found = true
        break

    foreach tetangga : currentNode.tetangga
        gn = currentNode.Route.Sum
        hn = tetangga.distanceToGoal
        if tetangga is visited then skip
        else
            Update tetangga.lowest_cost = gn
            Rute = currentNode.rute + tetangga
            PQ.enqueue(gn + hn, tetangga, Rute)

    if found then return Rute
else return []
```

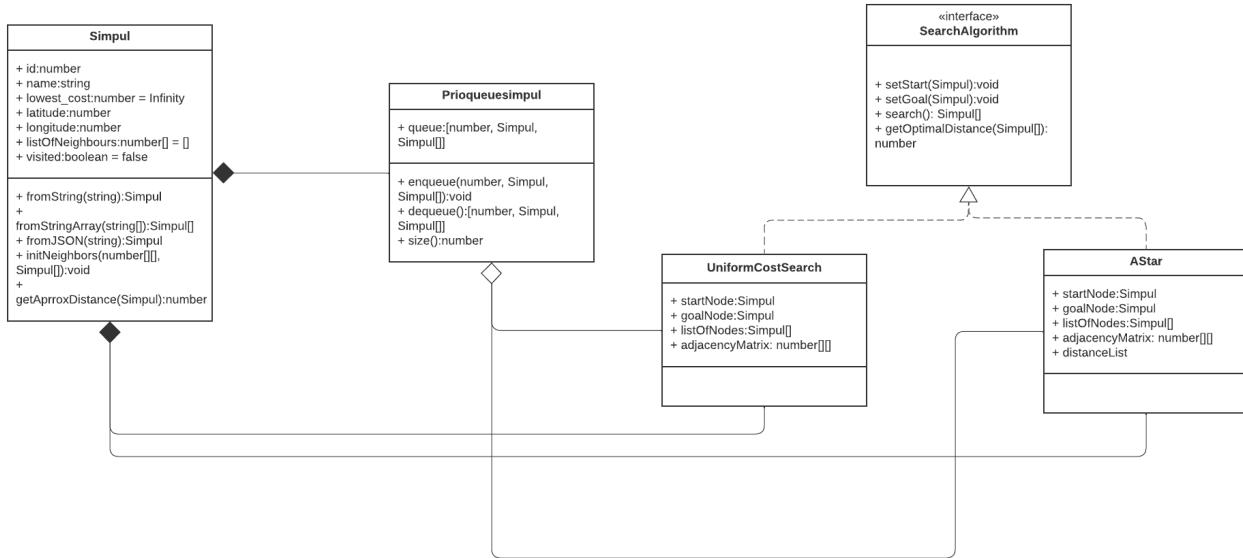
BAB III

RANCANGAN DAN IMPLEMENTASI PROGRAM

3.1 Rancangan Algoritma

Pada tugas ini, kami merancang algoritma kami dalam bahasa TypeScript, dan menggunakan Pemrograman Berorientasi Objek, berikut adalah UML Diagram rancangan kami:

Gambar 3.1.1 Diagram UML



Pada Diagram UML kami, kami membuat sebuah kelas Simpul yang merepresentasikan simpul-simpul lokasi yang terdapat pada peta beserta atribut-atributnya, selanjutnya ada juga kelas Prioqueuesimpul yang adalah sebuah *priority queue* yang berisikan *priority number*, simpul, dan juga rute untuk ke simpul tersebut. Kedua kelas tersebut digunakan oleh kelas algoritma UniformCostSearch dan AStar yang mengimplementasikan sebuah *interface* SearchAlgorithm. Berikut adalah program lengkap dari setiap bagian:

Simpul.ts

```

import L from "leaflet"

class Simpul {
    id: number;
    name: string;
    lowest_cost: number = Infinity;
    latitude: number;
    longitude: number;
}
  
```

```
// List of neighbors is the distance of neighbors in the adjacency
matrix

// The index + 1 is the neighbor's id
listOfNeighbours: number[] = [];
visited: boolean = false;

constructor(id: number, name: string, latitude: number, longitude:
number) {
    this.id = id;
    this.name = name;
    this.latitude = latitude;
    this.longitude = longitude;
}

setLatitude(latitude: number) {
    this.latitude = latitude;
}

setLongitude(longitude: number) {
    this.longitude = longitude;
}

toString(): String {
    return `id: ${this.id}, name: ${this.name}, lowest_cost:
${this.lowest_cost}, latitude: ${this.latitude}, longitude:
${this.longitude}`;
}

static fromString(str: string): Simpul {
    const [id, name, latitude, longitude] = str.split(',');
    return new Simpul(Number(id), String(name), Number(latitude),
Number(longitude));
}

static fromStringArray(strArray: string[]): Simpul[] {
    return strArray.map(str => Simpul.fromString(str));
}

static fromJSON(json: string): Simpul {
```

```

        return Simpul.fromJson(json);
    }

initNeighbors(adjacencyMatrix : Number[][][], nodes : Simpul[]) {
    var neighbors = adjacencyMatrix[this.id - 1];
    for (var i = 0; i < neighbors.length; i++) {
        this.listOfNeighbours.push(Number(neighbors[i])) === 0 ?
Infinity : this.getApproxDistance(nodes[i]));
    }
}

getApproxDistance(other: Simpul): number {
    let point1 = new L.LatLng(this.latitude, this.longitude);
    let point2 = new L.LatLng(other.latitude, other.longitude);
    return point1.distanceTo(point2);
}
}

export {Simpul}

```

Prioqueuesimpul.ts

```

import { Simpul } from "./Simpul";

class Prioqueuesimpul {
    queue: [number, Simpul, Simpul[][][]];

    constructor() {
        this.queue = [];
    }

    enqueue(prio: number, element: Simpul, listPred: Simpul[]): void {
        let newArray = [...listPred]
        if (this.queue.length == 0) {
            this.queue.push([prio, element, newArray]);
            return;
        }
        for (let i = 0; i < this.queue.length; i++) {

```

```

        if (this.queue[i][0] > prio) {
            this.queue.splice(i, 0, [prio, element, newArray]);
            return;
        } else if(i == this.queue.length - 1) {
            this.queue.push([prio, element, newArray]);
            return;
        }
    }

}

dequeue(): [number, Simpul, Simpul[]] | undefined {
    return this.queue.shift();
}

size(): number {
    return this.queue.length;
}

export { Prioqueuesimpul }

```

SearchAlgorithm.ts

```

import { Simpul } from "./Simpul";

export interface SearchAlgorithm {
    setStart(start: Simpul): void;
    setGoal(goal: Simpul): void;
    search(): Simpul[];
    getOptimalDistance(route: Simpul[]): number;
}

```

UniformCostSearch.ts

```

import { Simpul } from "./Simpul"

```

```

import { SearchAlgorithm } from "./SearchAlgorithm";
import { Prioqueuesimpul } from "./Prioqueuesimpul";

class UniformCostSearch implements SearchAlgorithm {
    startNode: Simpul;
    goalNode: Simpul;
    listofNodes: Simpul[];
    adjacencyMatrix: Number[][];

    constructor(startNode: Simpul, goalNode: Simpul, nodes: Simpul[], matrix: number[][]) {
        this.startNode = startNode;
        this.goalNode = goalNode;
        this.adjacencyMatrix = matrix;
        this.listofNodes = [];
        for (let i = 0; i < nodes.length; i++) {
            this.listofNodes.push(new Simpul(nodes[i].id, nodes[i].name, nodes[i].latitude, nodes[i].longitude));
        }
        for (let i = 0; i < nodes.length; i++) {
            this.listofNodes[i].initNeighbors(this.adjacencyMatrix, this.listofNodes);
        }
    }

    setStart(start: Simpul): void {
        this.startNode = start;
    }

    setGoal(goal: Simpul): void {
        this.goalNode = goal;
    }

    search(): Simpul[] {
        // UCS algorithm
        // Start from start node

        this.listofNodes[this.startNode.id - 1].lowest_cost = 0;
        this.startNode.lowest_cost = 0;
    }
}

```

```

let openList = new Prioqueuesimpul();
openList.enqueue(0, this.startNode, []);

let current;
let currentNode;
let currentCost;
let currentRoute;
let found = false;

while (openList.size() > 0 && !found) {
    current = openList.dequeue(); // @ts-ignore
    currentCost = current[0]; // @ts-ignore
    currentNode = current[1]; // @ts-ignore
    currentRoute = current[2];
    this.listofNodes[currentNode.id - 1].visited = true;
    found = currentNode.id === this.goalNode.id;
    for (let i = 0; i < currentNode.listOfNeighbours.length &&
!found; i++) {
        // Assign the cost, if higher than the current cost and
        visited, then skip (no need to enqueue the route)
        if (!Number.isFinite(currentNode.listOfNeighbours[i])) {
            continue;
        }
        let neighborDistance = currentNode.listOfNeighbours[i];
        let neighborIdx = i;
        // neighbor id is idx + 1
        if (this.listofNodes[neighborIdx].lowest_cost <=
neighborDistance + currentCost && this.listofNodes[neighborIdx].visited)
        {
            continue;
        } else {
            // update lowest cost if lower
            if (this.listofNodes[neighborIdx].lowest_cost >
neighborDistance + currentCost) {
                this.listofNodes[neighborIdx].lowest_cost =
neighborDistance + currentCost;
            }
            let newRoute = [...currentRoute];

```

```

        newRoute.push(currentNode);
        openList.enqueue(neighborDistance + currentCost,
this.listofNodes[neighborIdx], newRoute);
    }
}

// Check if found or not
let route: Simpul[] = [];
if (found) { // @ts-ignore
    route = currentRoute; // @ts-ignore
    route.push(currentNode);
}

return route;
}

getOptimalDistance(route: Simpul[]): number {
    let ret = 0;
    for (let i = 0; i < route.length - 1; i++) {
        ret = ret + route[i].getApproxDistance(route[i+1]);
    }
    return ret / 1000;
}

export { UniformCostSearch }

```

AStar.ts

```

import { Simpul } from "./Simpul";
import { SearchAlgorithm } from "./SearchAlgorithm";
import { PrioqueueSimpul } from "./PrioqueueSimpul";

class AStar implements SearchAlgorithm {
    startNode: Simpul;
    goalNode: Simpul;
    listofNodes: Simpul[];
}

```

```

adjacencyMatrix: Number[][];

// Distance Matrix computes the approx. distance from all nodes to
the goal node

distanceList: Number[];

constructor(startNode: Simpul, goalNode: Simpul, nodes: Simpul[], matrix: number[][]) {
    this.startNode = startNode;
    this.goalNode = goalNode;
    this.adjacencyMatrix = matrix;
    this.listofNodes = [];
    this.distanceList = [];
    for (let i = 0; i < nodes.length; i++) {
        this.listofNodes.push(new Simpul(nodes[i].id, nodes[i].name,
nodes[i].latitude, nodes[i].longitude));
    }

    for (let i = 0; i < nodes.length; i++) {
        this.listofNodes[i].initNeighbors(this.adjacencyMatrix,
this.listofNodes);
    }

    for (let i = 0; i < this.listofNodes.length; i++) {

this.distanceList.push(this.listofNodes[i].getApproxDistance(goalNode));
    }
}

setStart(start: Simpul): void {
    this.startNode = start;
}

setGoal(goal: Simpul): void {
    this.goalNode = goal;
}

search(): Simpul[] {
    // A* Algorithm
}

```

```

// Start from start node

this.listofNodes[this.startNode.id - 1].lowest_cost = 0;
this.startNode.lowest_cost = 0;

let openList = new Prioqueuesimpul();
openList.enqueue(Number(this.distanceList[this.startNode.id - 1]), this.startNode, [this.startNode]);

let current;
let currentNode;
let currentCost;
let currentRoute;
let found = false;
while (openList.size() > 0 && !found) {
    current = openList.dequeue();
    // Current Cost is Current Priority (f(n))
    // @ts-ignore
    currentCost = current[0]; // @ts-ignore
    currentNode = current[1]; // @ts-ignore
    currentRoute = current[2];
    this.listofNodes[currentNode.id - 1].visited = true;
    found = currentNode.id === this.goalNode.id;
    for (let i = 0; i < currentNode.listOfNeighbours.length && !found; i++) {
        // No need to visit if already visited
        if (!Number.isFinite(currentNode.listOfNeighbours[i])) {
            continue;
        }
        let neighborDistance = currentNode.listOfNeighbours[i];
        let neighborIdx = i;
        // neighbor id is idx + 1
        let nextRoute = [...currentRoute];
        nextRoute.push(this.listofNodes[neighborIdx]);
        // Get g(n)
        let gn: number = 0;
        for (let j = 0; j < nextRoute.length - 1; j++) {
            gn +=
```

```

nextRoute[j].listOfNeighbours[nextRoute[j+1].id-1]
    }
    // Get h(n)
    let hn: number = Number(this.distanceList[neighborIdx]);
    // No need to enqueue if already visited
    if (this.listofNodes[neighborIdx].visited) {
        continue;
    } else {
        // update lowest cost if lower, only updates to g(n)
        if (this.listofNodes[neighborIdx].lowest_cost > gn)
    {
        this.listofNodes[neighborIdx].lowest_cost = gn;
    }
    openList.enqueue(gn + hn,
this.listofNodes[neighborIdx], nextRoute);
    }
}
}

// Check if found or not
let route: Simpul[] = []
if (found) { // @ts-ignore
    route = currentRoute;
    // route.push(currentNode);
}

return route;
}

getOptimalDistance(route: Simpul[]): number {
    let ret = 0;
    for (let i = 0; i < route.length - 1; i++) {
        ret = ret + route[i].getApproxDistance(route[i+1]);
    }
    return ret / 1000;
    // Technically can only return goalNode.lowest_cost
}
}

```

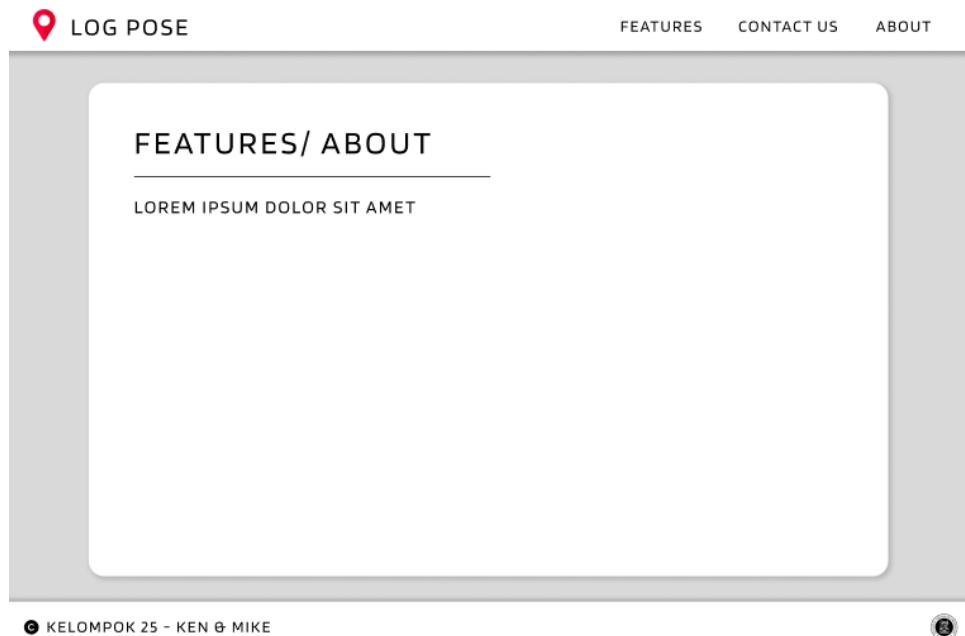
```
export { AStar }
```

3.2 Rancangan Aplikasi

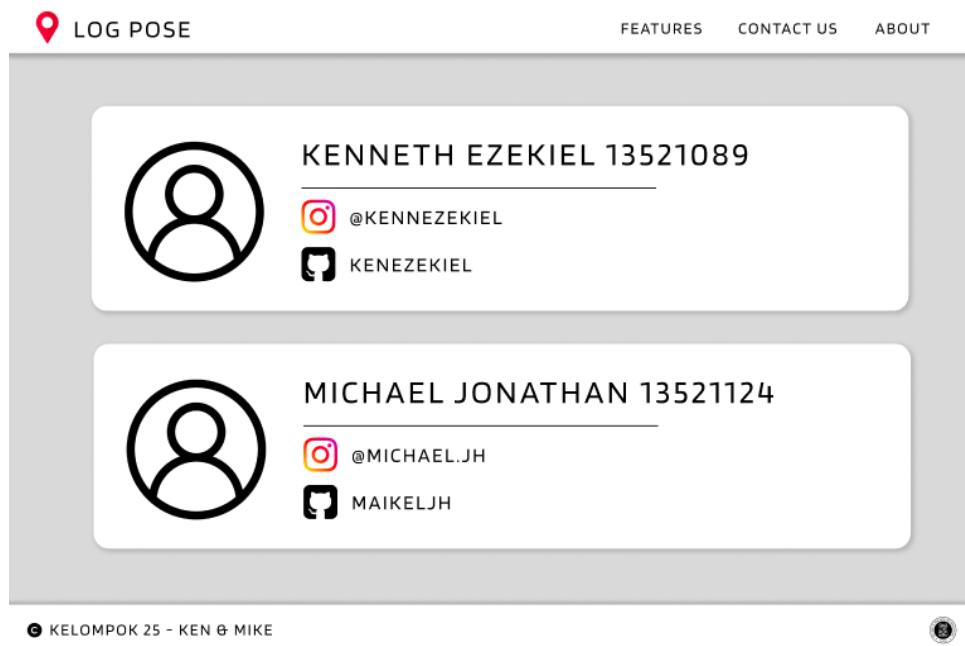
Pada tugas ini, kami merancang aplikasi kami menggunakan web application agar aplikasi dapat mudah di-deploy sehingga dapat digunakan oleh masyarakat umum. Framework yang digunakan dalam pengembangan aplikasi web ini adalah Next.js sebagai framework dari typescript dan Tailwind sebagai framework dari CSS. Untuk mempermudah pengembangan web ini, kami membuat rancangan desain kasarnya terlebih dahulu untuk mendapat gambaran tampilan websitenya akan seperti apa. Berikut adalah beberapa hasil desain kasar aplikasi web kami.



Gambar 3.2.1 Desain Tampilan Utama Aplikasi



Gambar 3.2.2 Desain Tampilan About Aplikasi

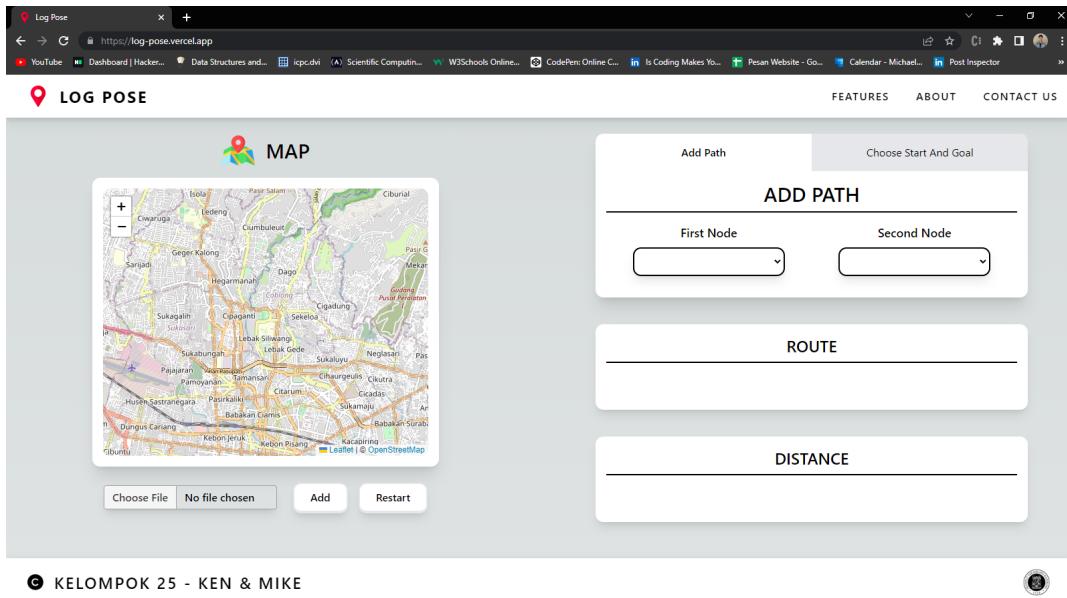


Gambar 3.2.3 Desain Tampilan Contact Us Aplikasi

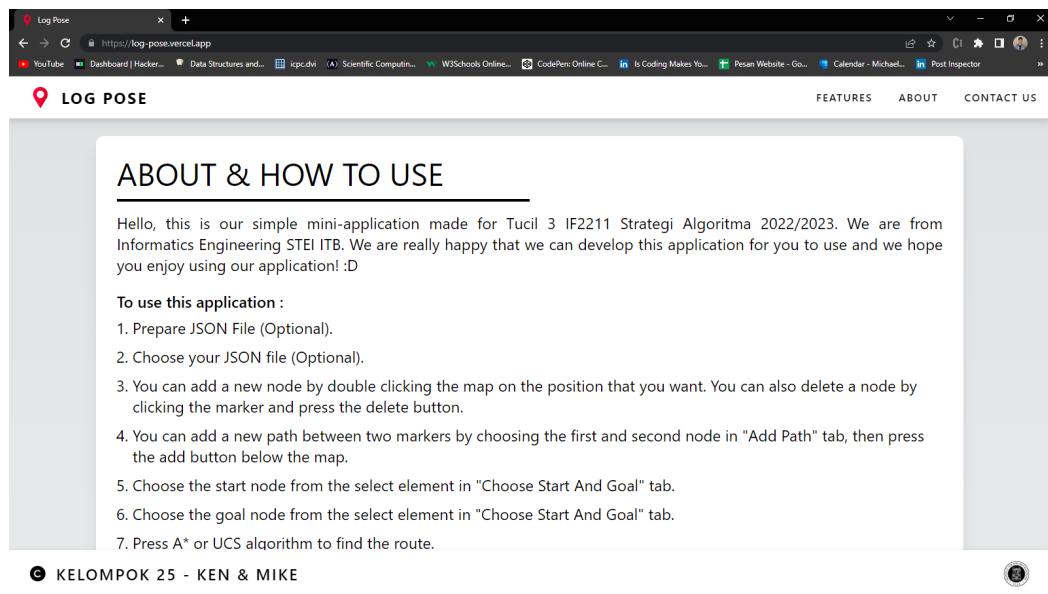
Setelah selesai merancang desain aplikasi, kami melakukan setup project webnya dengan node package manager. Untuk kode keseluruhan dari website, dapat dilihat pada pranala Github yang diterterakan pada lampiran. Karena kami mengimplementasikan bonus, terdapat beberapa aspek yang harus diperhatikan saat pembuatan UI dari website, yaitu

1. Karena tampilan yang dibuat tidak memiliki dimensi yang *fix*, maka diperlukan pembuatan tampilan yang responsif sehingga tampilan tetap terlihat bagus dan nyaman untuk berbagai device yang digunakan.
2. Tampilan seharusnya tidak berbeda jauh dengan yang sudah dirancang pada desain sebelumnya dan kami berhasil menyesuaikan dengan desain walaupun tidak sama persis.
3. Karena pada spek tertera bahwa pengguna harus bisa berinteraksi dengan peta, kami menambahkan beberapa fitur yang dapat digunakan oleh pengguna pada peta, yakni
 - a. Pengguna dapat menambahkan simpul dengan melakukan double click pada lokasi yang ingin dibuat menjadi simpul.
 - b. Pengguna dapat menambahkan *path* pada tab “Add Path” dengan memilih node pertama dan node kedua yang ingin dihubungkan. Arah dari path akan muncul jika penambahan path dilakukan dari node pertama ke node kedua saja. Jika dari node kedua ke node pertama ditambahkan path juga, maka arah dari path akan hilang yang menandakan bahwa path dapat dilalui secara dua arah.
 - c. Pengguna dapat menggunakan test-case nya sendiri dengan mengupload file pada website dalam format .json dan sudah diberikan contoh pula pada *section how to use*.
 - d. Pengguna dapat memilih *start node* dan *goal node* pada tab “Choose Start And Goal”. Kemudian, pengguna dapat mengklik tombol A* untuk melakukan pencarian rute secara algoritma A* dan tombol UCS untuk melakukan pencarian rute secara algoritma UCS.
 - e. Pengguna dapat melihat nama simpul dan juga bobot dari path (jarak) dengan mengklik simpul/path yang ingin dilihat informasinya.
 - f. Pengguna dapat menghapus simpul/path dengan mengklik elemen tersebut dan mengklik tombol “Delete” di dalam konten *popup* yang tertampil.
 - g. Pengguna dapat memindahkan simpul yang sudah dibuat dengan men-drag simpul tersebut ke lokasi lain.
 - h. Algoritma UCS dan A* yang dipilih sebelumnya akan dieksekusi kembali jika pengguna berinteraksi dengan simpul atau path pada peta sehingga rute terpendek yang dihasilkan tetap rute terpendek dari graf yang sudah di-update berdasarkan interaksi user.
 - i. Jika pengguna ingin melakukan restart, pengguna dapat mengklik tombol “Restart” di bawah peta untuk menghapus seluruh simpul dan path yang sudah dibentuk pada peta.
4. Ditampilkan *popup message* untuk menandakan apakah *file* yang di-upload pengguna valid atau tidak, algoritma pencarian rute berhasil atau tidak, penambahan juga penghapusan simpul dan path, dan *restart* peta.
5. Dibuatkan juga section untuk cara menggunakan aplikasi sehingga memudahkan pengguna awam untuk mengetahui cara menggunakan aplikasi web ini.

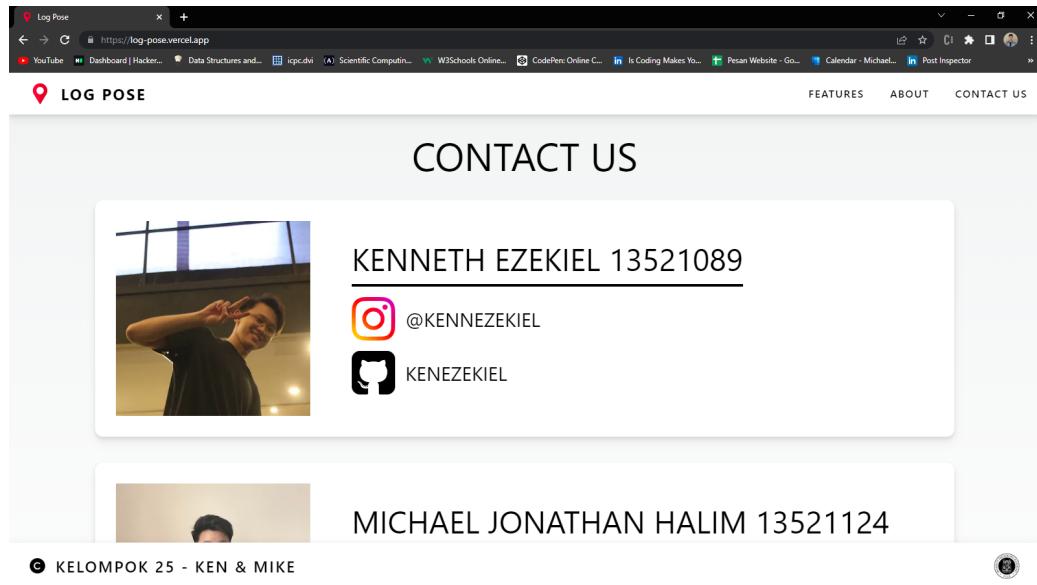
Berikut adalah tampilan akhir dari aplikasi website “Log Pose” ini.



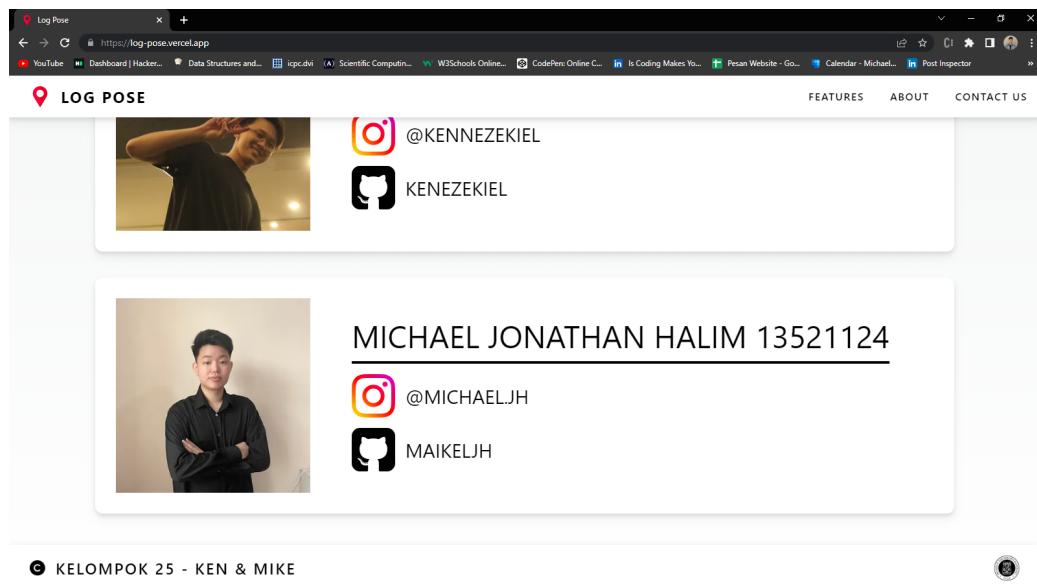
Gambar 3.2.4 Tampilan Utama Aplikasi Log Pose



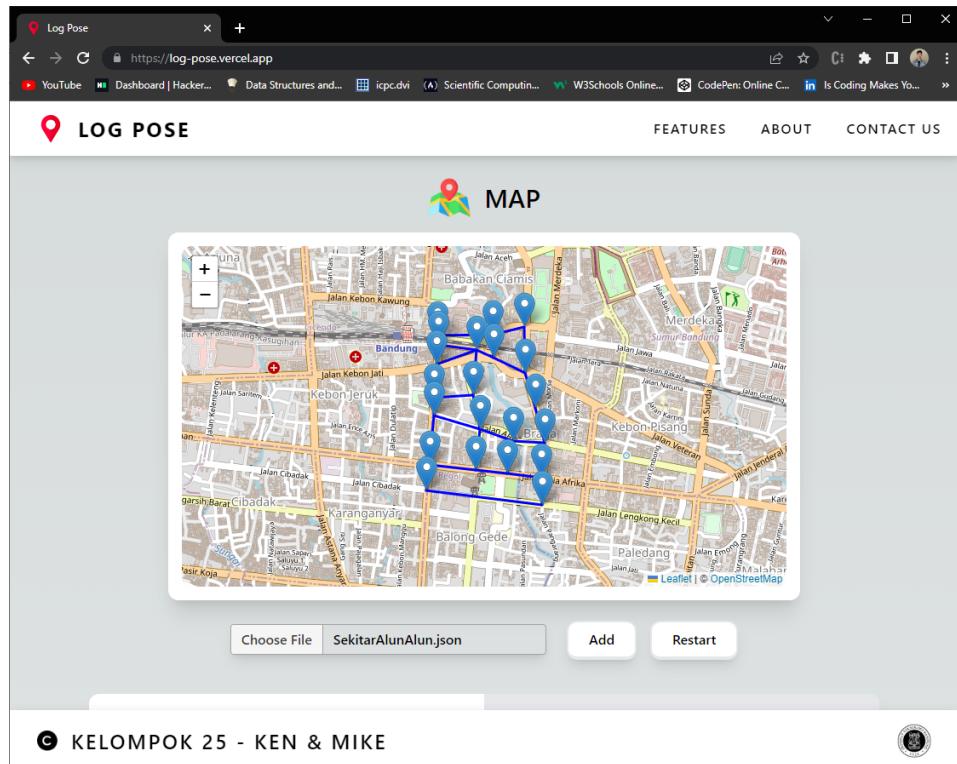
Gambar 3.2.5 Tampilan About & How To Use Aplikasi Log Pose



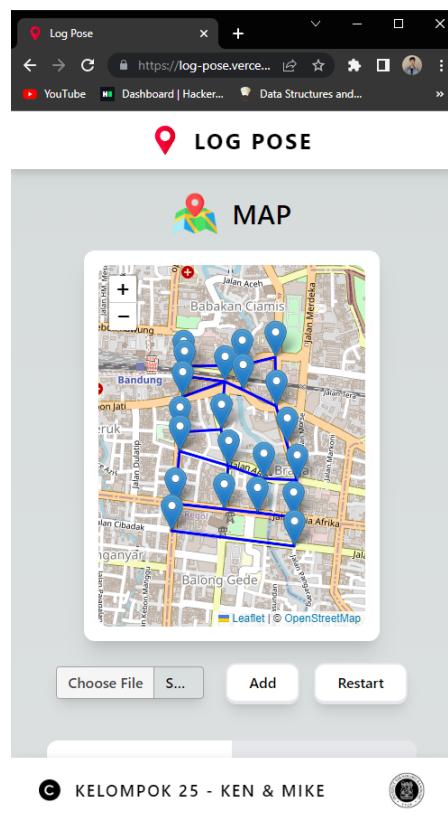
Gambar 3.2.6 Tampilan About Us Log Pose



Gambar 3.2.7 Tampilan About Us Lanjutan Aplikasi Log Pose



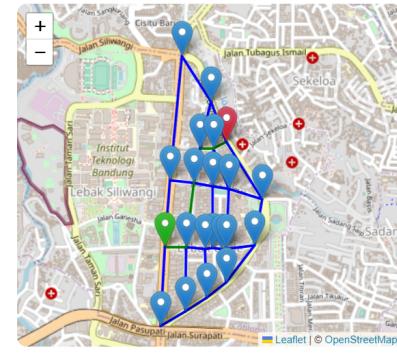
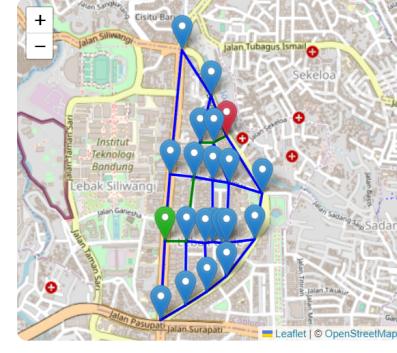
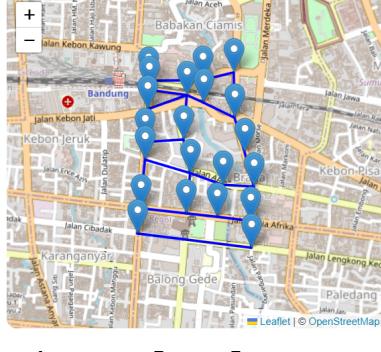
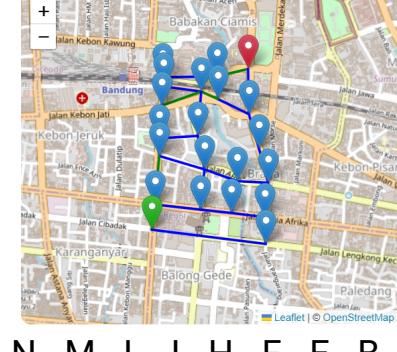
Gambar 3.2.8 Tampilan Utama Aplikasi Log Pose Dengan Dimensi Tablet

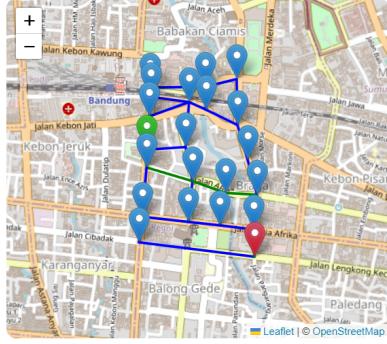
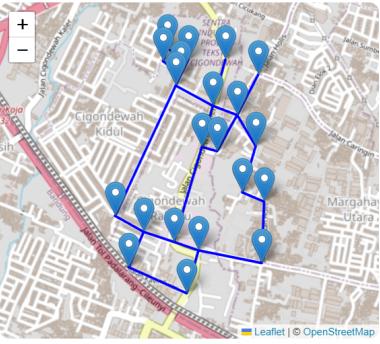
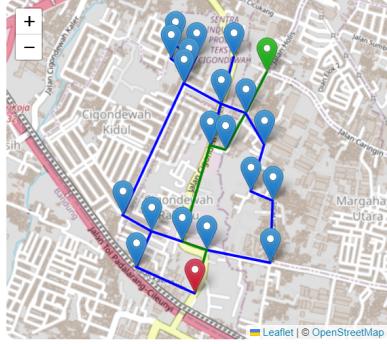
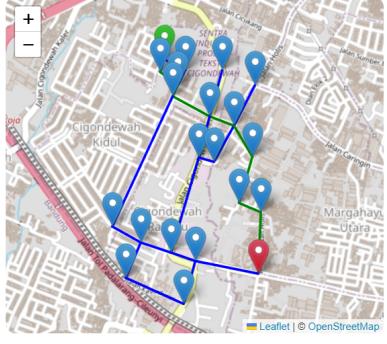


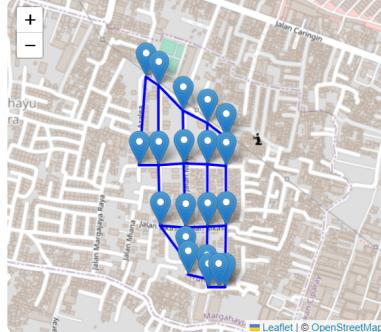
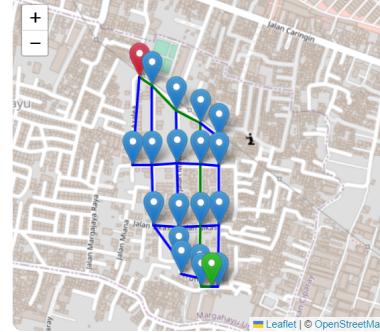
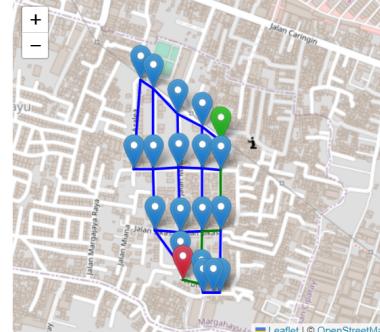
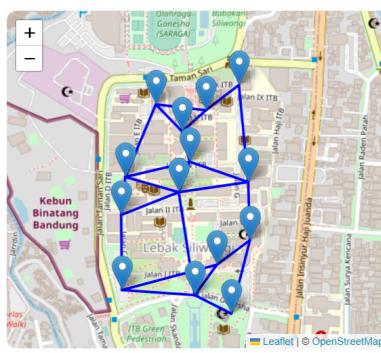
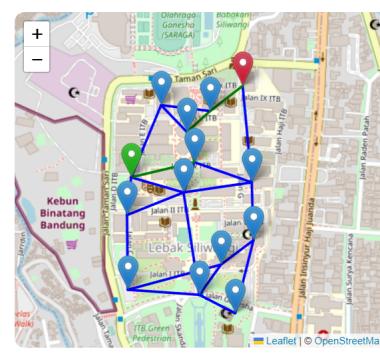
Gambar 3.2.9 Tampilan Utama Aplikasi Log Pose Dengan Dimensi Smartphone

BAB IV

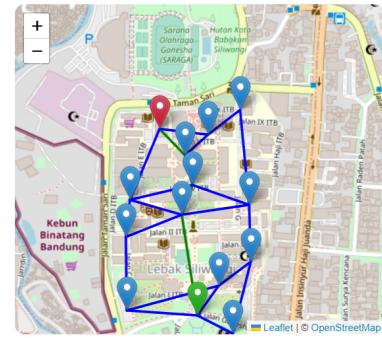
HASIL PENGUJIAN

Test Case	Algoritma : Goal - Start	Rute - Jarak
 RSBorromeus.json	A* E : RS - S	 E : RS - F - M - Q - R - S 0.82655 km
	UCS E : RS - S	 E : RS - F - M - Q - R - S 0.82655 km
 SekitarAlunAlun.json	A* N - C	 N - M - L - I - H - F - E - B - C 1.35124 km

	<p>UCS I - U</p>	 <p>I - L - K - Q - R - T - U 1.08859 km</p>
 <p>BandungSelatanSMAKen.json</p>	<p>A* I - Q</p>	 <p>I - H - K - J - T - P - Q 1.53588 km</p>
	<p>UCS A - O</p>	 <p>A - B - D - E - F - H - L - M - N - O 1.61177 km</p>

 <p>Pasadena.json (Daerah Rumah)</p>	<p>A* Rumah - K</p>	 <p>Rumah - R - P - D - J - I - A - H - K 0.70800 km</p>
<p>UCS M - Q</p>		 <p>M - N - O - D - P - Q 0.49958 km</p>
 <p>ITB.json</p>	<p>A* GKU Barat - CRCS</p>	 <p>GKU Barat - Plazwid - Oktagon - Perpustakaan Pusat - CRCS 0.47727 km</p>

UCS Kubus - CADL



Kubus - Labtek V -
Plazwid - Oktagon -
CADL
0.57749 km

LOG POSE

MAP

File tidak valid!

Add Path Choose Start And Goal

Start Node Goal Node

ROUTE

DISTANCE

Choose File NotVa...-1.json A* UCS Restart

© KELOMPOK 25 - KEN & MIKE

NotValid-1.json

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Pada tugas kecil III IF2211 Strategi Algoritma ini telah diimplementasikan algoritma pencarian rute dengan algoritma *Uniform Cost Search* dan *A** beserta fungsi-fungsi pendukung dalam tujuan untuk menyelesaikan pencarian rute terpendek dalam graf berbobot. Fungsi-fungsi tersebut mencakup fungsi yang menangani masukan simpul awal dan tujuan, peta, dan juga sebuah *priority queue* untuk mengatur urutan penjelajahan. Implementasi tersebut kemudian berhasil direalisasikan dalam sebuah aplikasi berbasis web dengan bahasa Typescript.

Algoritma *Uniform Cost Search* pada tugas ini digunakan untuk penyelesaian masalah pencarian rute terdekat, dimana akan ditentukan dari himpunan simpul yang bertetangga, mana simpul berikutnya yang akan dijelajahi, menggunakan sebuah fungsi penentu.

Dengan demikian, penulis menyimpulkan bahwa melalui Tugas Kecil III IF2211 Strategi Algoritma ini, dapat dibuat sebuah algoritma pencarian rute terdekat menggunakan algoritma *Uniform Cost Search* dan *A** jika diberikan sebuah graf berbobot dengan N buah simpul.

5.2. Saran

Tugas Kecil IF2211 Strategi Algoritma Semester II Tahun 2022/2023 menjadi salah satu tugas yang memberikan pelajaran baru bagi penulis. Berdasarkan pengalaman penulis mengerjakan tugas ini, berikut merupakan saran untuk pembaca yang ingin melakukan atau mengerjakan hal yang serupa.

1. Keefektifan dalam kerjasama tim merupakan hal yang penting dalam mengerjakan tugas ini. Tugas ini sangat terbantu oleh pemakaian *real-time collaboration app*. Selain itu, pemakaian aplikasi pengelola version control seperti Github sangat disarankan agar memudahkan untuk mengelola pekerjaan secara asinkron.
2. Dibutuhkan sebuah list dari *dependencies* dan *library* yang akan digunakan. Tugas ini berbasis *web* dan sangat terbantu dengan pemakaian *library*, tetapi, *library* tersebut menjadi sebuah *dependency* yang kemudian perlu di-*install* untuk menjalankan *web* secara lokal.
3. Pemahaman Struktur Data, Algoritma *A** dan *Uniform Cost Search*, dan Konsep Pemrograman Berorientasi Objek sangat penting dalam penggerjaan tugas. Dengan pemahaman yang memadai, penulis merasa sangat dimudahkan dalam perancangan sampai implementasi algoritma.
4. Pemahaman *web development* merupakan hal yang sangat membantu dalam penggerjaan tugas ini. Hal tersebut dikarenakan Bonus yang diberikan, yaitu penggunaan API sangat dimudahkan jika aplikasi dapat dibuat berbasis *web*.

DAFTAR PUSTAKA

Maulidevi, Nur Ulfa. 2022. Penentuan Rute 1. Bandung: Institut Teknologi Bandung.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

Maulidevi, Nur Ulfa. 2022. Penentuan Rute 2. Bandung: Institut Teknologi Bandung.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>

Ekete, David. 2022. Object-Oriented Programming in TypeScript.
<https://birdeatsbug.com/blog/object-oriented-programming-in-typescript>

Irovich, Gleb. 2020. Stack and Queue in TypeScript.
<https://dev.to/glebirovich/typescript-data-structures-stack-and-queue-hld>

LAMPIRAN

Pranala Repository GitHub: github.com/maikeljh/Tucil3_13521089_13521124

Pranala Deployment Website: <https://log-pose.vercel.app/>

Pranala Desain Website:

<https://www.figma.com/file/i2DQRI2xrVI9l80exgUUtO/Tucil-3?node-id=0%3A1&t=jxoAHW9yqjdXQYWV-1>

No	Keterangan	Done
1	Program dapat menerima input graf	<input checked="" type="checkbox"/>
2	Program dapat menghitung lintasan terpendek dengan UCS	<input checked="" type="checkbox"/>
3	Program dapat menghitung lintasan terpendek dengan A*	<input checked="" type="checkbox"/>
4	Program dapat menampilkan lintasan terpendek serta jaraknya	<input checked="" type="checkbox"/>
5	Bonus: Program dapat menerima input peta dengan Map API dan menampilkan peta serta lintasan terpendek pada peta	<input checked="" type="checkbox"/>