

TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek



Semester II Tahun 2022/2023

Disusun Oleh:

13521114 Farhan Nabil S.

13521123 William Nixon

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

Bab I

Deskripsi Persoalan

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.

Bab II

Deskripsi Algoritma

1. Algoritma UCS

Algoritma UCS adalah salah satu algoritma pencarian tanpa informasi pada graf. UCS bekerja dengan cara melakukan pencarian berurutan dengan simpul yang jarak akumulatif (*cost*) dari simpul asal terendah. sisi. Algoritma ini menggunakan strategi pencarian berurutan (best-first search) dengan memilih simpul yang memiliki nilai cost (biaya) terkecil dari simpul asal ke simpul tersebut.

Dengan menggunakan strategi pencarian berurutan dan memilih simpul dengan cost terkecil, UCS dapat menjamin bahwa jalur yang ditemukan merupakan jalur terpendek dari simpul asal ke simpul tujuan. Namun, UCS juga dapat mengalami masalah eksponensial dengan jumlah simpul yang besar atau jika terdapat sisi dengan cost negatif. Oleh karena itu, algoritma ini sering digunakan dalam lingkup yang terbatas atau jika sisi dengan cost negatif tidak mungkin ada.

2. Algoritma A*

Algoritma A* adalah salah satu algoritma pencarian yang *informed* pada graf. A* bekerja seperti UCS namun *cost* pada A* ditambah dengan suatu fungsi heuristik. Hal ini bertujuan agar pencarian menghindari simpul yang menjauh dari tujuan. Agar A* dapat menjamin hasil yang diberikan merupakan jarak terpendek, hasil dari fungsi heuristik harus selalu kurang dari sama dengan jarak aslinya.

Dengan menggunakan fungsi heuristik, algoritma A* dapat menemukan jalur terpendek dengan lebih efisien daripada algoritma UCS dalam situasi di mana simpul yang mencapai tujuan lebih jauh dari simpul awal. Namun, seperti halnya algoritma UCS, algoritma A* juga dapat mengalami masalah eksponensial dengan jumlah simpul yang besar atau jika fungsi heuristik tidak memenuhi syarat bahwa nilai fungsi heuristik selalu kurang dari atau sama dengan jarak aslinya.

3. Implementasi

Pada program kami, kami mengimplementasikan UCS dengan A* menggunakan *priority queue* yang memprioritaskan *cost* terkecil. *Cost* dihitung berdasarkan jumlah bobot yang telah ditempuh untuk mencapai simpul tersebut pada UCS. Pada A*, *cost* ini dijumlahkan dengan fungsi heuristik berupa jarak nyata hasil perhitungan antara koordinat simpul asal dan simpul tujuan. Kami juga mengabaikan simpul sebelum setiap menambahkan ke *queue* karena kembali ke simpul sebelum pasti bukan *shortest path*.

Langkah-langkah dari implementasi kami adalah:

1. Inisialisasi: Membuat Priority Queue yang hanya berisi node start
2. Pilih simpul dengan cost terkecil dari simpul asal yang sedang dieksplorasi. Jika simpul tersebut adalah simpul tujuan, maka algoritma akan berhenti dan mengembalikan langkah-langkahnya. Jika tidak, simpul tersebut akan di-*mark* sebagai *visited*.
3. Untuk setiap simpul yang terhubung dengan simpul yang sedang dieksplorasi, akan dilakukan hal sebagai berikut
 - a. Jika simpul yang akan dieksplorasi sudah *visited*, simpul akan diabaikan.
 - b. Hitung cost baru untuk mencapai simpul tersebut melalui simpul yang sedang dieksplorasi. Pada UCS, cost bernilai hasil jumlah cost saat ini dan jarak antara simpul yang sedang dieksplorasi dan simpul yang akan dieksplorasi. Pada A*, cost akan ditambah dengan nilai heuristik yaitu jarak lurus antara kedua simpul
 - c. Simpul berikutnya akan di-*enqueue* pada Priority Queue
4. Ulangi langkah 2 dan 3 hingga simpul tujuan ditemukan atau *queue* kosong.

Hasil uji coba kami menunjukkan bahwa mayoritas test case akan menghasilkan hasil dan rute yang sama dengan kedua metode. Hal ini karena test case kami dibuat dengan baik, karena memenuhi syarat pada algoritma A* yaitu “hasil dari fungsi heuristik harus selalu kurang dari sama dengan jarak aslinya.” Hasil heuristik dan jarak antar simpul pada mode bonus dihitung secara penuh melalui jarak antar latitude dan longitude simpul, sedangkan pada mode non-bonus, hasil heuristik diperoleh dari jarak latitude dan longitude, dan jarak antar simpul dari matriks berbobot.

Bab III

Source Code

```
class Node {
    // Class untuk Node
    constructor(lat, lng) {
        this.lat = lat;
        this.lng = lng;
        this.edges = [];
    }

    // Menambahkan Edge untuk Node ini
    addEdge(dest, distance) {
        this.edges.push(new Edge(this, dest, distance));
    }
}

class Edge {
    // Class untuk Edge
    // Hanya menyimpan data
    constructor(src, dest, distance) {
        this.src = src;
        this.dest = dest;
        this.distance = distance;
    }
}

class Graph {
    // Class Graf
    constructor() {
        this.nodes = {};
    }

    // Menambah Node dalam Graf
    addNode(name, lat = 0, lng = 0) {
        this.nodes[name] = new Node(lat, lng);
    }

    // Menambah Edge dengan Kalkulasi Real Distance
}
```

```

addEdge(src, dest, isOneWay = false) {
    var dist = calculateDistance(this.nodes[src], this.nodes[dest]);
    this.nodes[src].addEdge(dest, dist);
    if (!isOneWay) {
        this.nodes[dest].addEdge(src, dist);
    }
}

// Menambah Edge dengan Given Distance
addWeightedEdge(src, dest, dist, isOneWay = false) {
    this.nodes[src].addEdge(dest, dist);
    if (!isOneWay) {
        this.nodes[dest].addEdge(src, dist);
    }
}

// Melakukan search
// Bisa mode aStar dan UCS
search(src, dest, mode) {
    var pq = new PriorityQueue();
    var ret = [[], -1];
    var visited = new Map();

    var initData;
    if (mode === "aStar") {
        initData = new AStarData(
            src,
            [],
            0,
            calculateDistance(this.nodes[src], this.nodes[dest])
        );
    } else if (mode === "UCS") {
        initData = new UCSData(src, [], 0);
    }

    pq.enqueue(initData);
    while (!pq.isEmpty()) {
        var current = pq.dequeue();

```

```

        if (visited[current.name]) {
            continue;
        }
        visited[current.name] = true;

        if (current.name === dest) {
            ret = [current.prev.concat([current.name]),
current.totalDist];
            break;
        }

        for (var i = 0; i < this.nodes[current.name].edges.length;
i++) {
            var edge = this.nodes[current.name].edges[i];
            if (!visited[edge.dest]) {
                var newData;
                if (mode === "aStar") {
                    newData = new AStarData(
                        edge.dest,
                        current.prev.concat([current.name]),
                        current.totalDist + edge.distance,
                        calculateDistance(
                            this.nodes[edge.dest],
                            this.nodes[dest]
                        )
                    );
                } else if (mode === "UCS") {
                    newData = new UCSData(
                        edge.dest,
                        current.prev.concat([current.name]),
                        current.totalDist + edge.distance
                    );
                }
                pq.enqueue(newData);
            }
        }
    }

    return ret;
}

```

```
    }

}

class PriorityQueue {
    // Class untuk membuat Queue dengan Prioritas
    constructor() {
        this.queue = [];
    }

    // Mengembalikan apakah queue kosong
    isEmpty() {
        return this.queue.length === 0;
    }

    // Melakukan Enqueue data
    enqueue(data) {
        let left = 0;
        let right = this.queue.length - 1;
        let idx = -1;

        while (left <= right) {
            var mid = Math.floor((left + right) / 2);

            if (this.queue[mid].cost > data.cost) {
                idx = mid;
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }

        if (idx === -1) {
            this.queue.push(data);
        } else {
            this.queue.splice(idx, 0, data);
        }
    }

    // Melakukan Dequeue dan Mengembalikan Isi yang Dihapus
}
```

```

dequeue() {
    if (!this.isEmpty()) {
        return this.queue.shift();
    } else {
        return null;
    }
}

class UCSData {
    // Class menyimpan data UCS
    constructor(name, prev, totalDist) {
        this.name = name;
        this.prev = prev;
        this.totalDist = totalDist;
        this.cost = totalDist;
    }
}

class AStarData {
    // Class menyimpan data A*
    constructor(name, prev, totalDist, additionalCost) {
        this.name = name;
        this.prev = prev;
        this.totalDist = totalDist;
        this.cost = totalDist + additionalCost;
    }
}

function calculateDistance(node1, node2) {
    // Mengembalikan jarak dalam Km
    // Source:
    https://community.powerbi.com/t5/Desktop/How-to-calculate-lat-long-distance/td-p/1488227

    // Radius Bumi dalam Km
    const earthRadius = 6371;

    // Mengubah derajat menjadi radian
}

```

```
var lat1 = toRadian(node1.lat);
var long1 = toRadian(node1.lng);
var lat2 = toRadian(node2.lat);
var long2 = toRadian(node2.lng);

return (
    earthRadius *
    Math.acos(
        Math.sin(lat1) * Math.sin(lat2) +
        Math.cos(lat1) * Math.cos(lat2) * Math.cos(long2 -
long1)
    )
);
}

function toRadian(degree) {
    // Fungsi mengubah derajat menjadi radian
    return (degree * Math.PI) / 180;
}

module.exports = Graph;
```

Bab IV

Hasil Uji

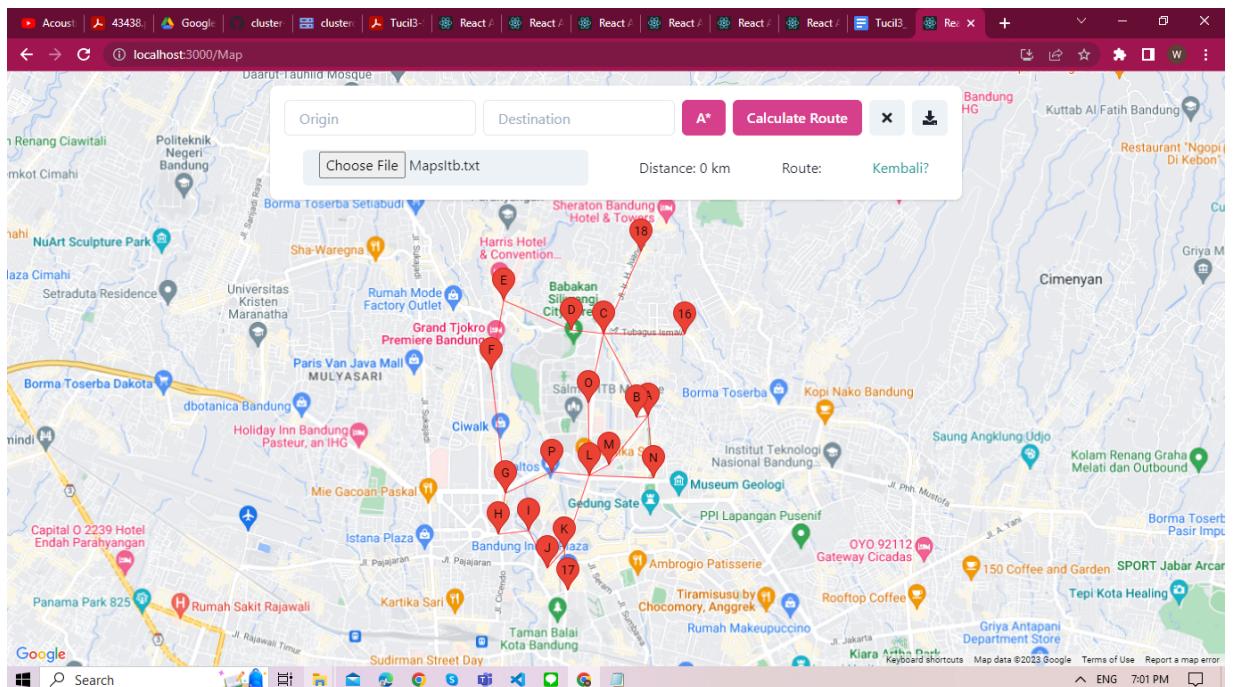
Diuji dengan ASUS Vivobook dengan spesifikasi:

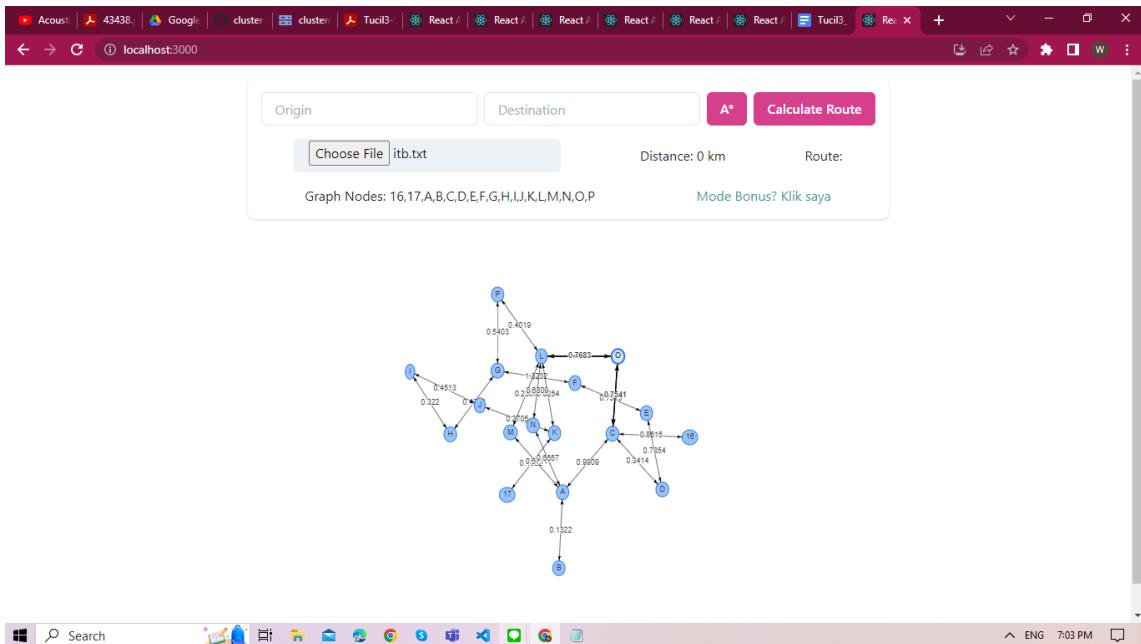
1. RAM 4 GB
 2. Prosesor Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz

1. Peta Kampus ITB

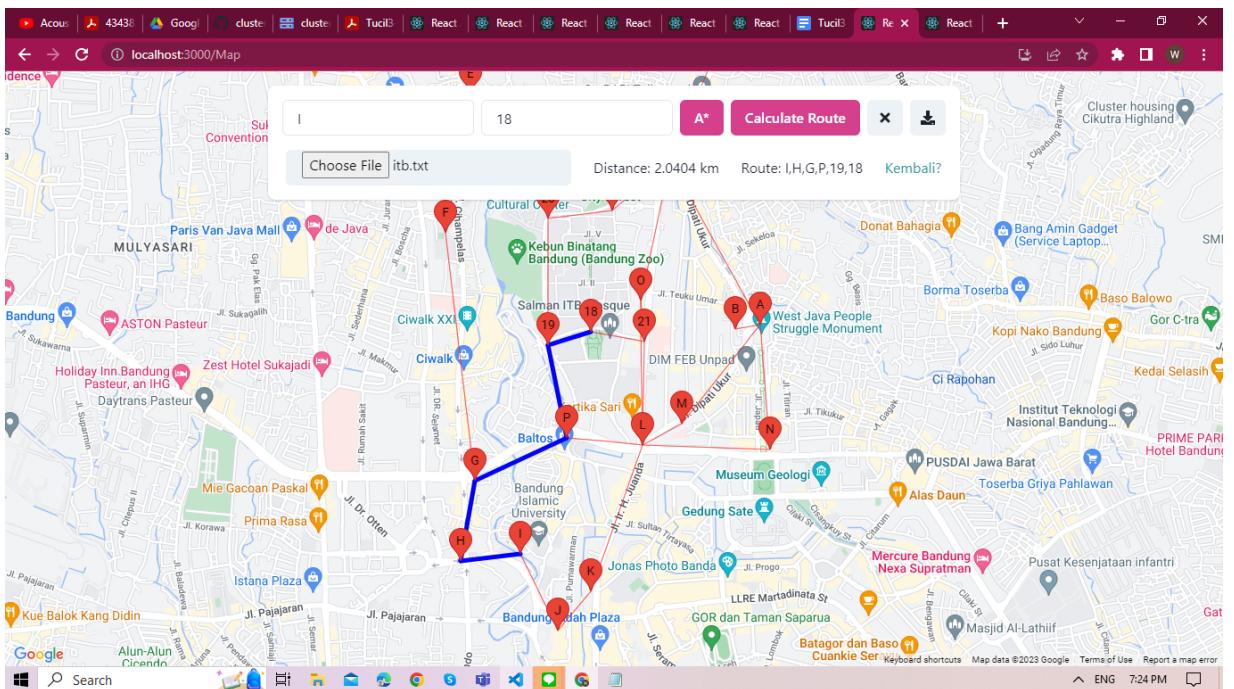
a. Input: itb.txt

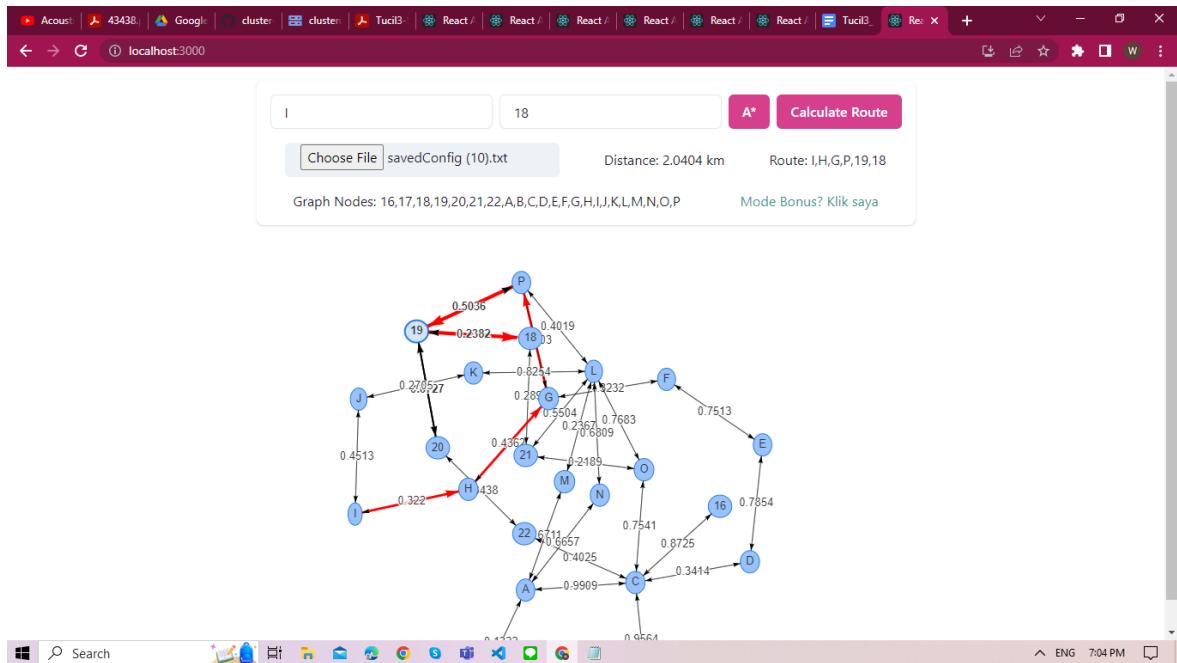
b. Tampilan Peta Inisial



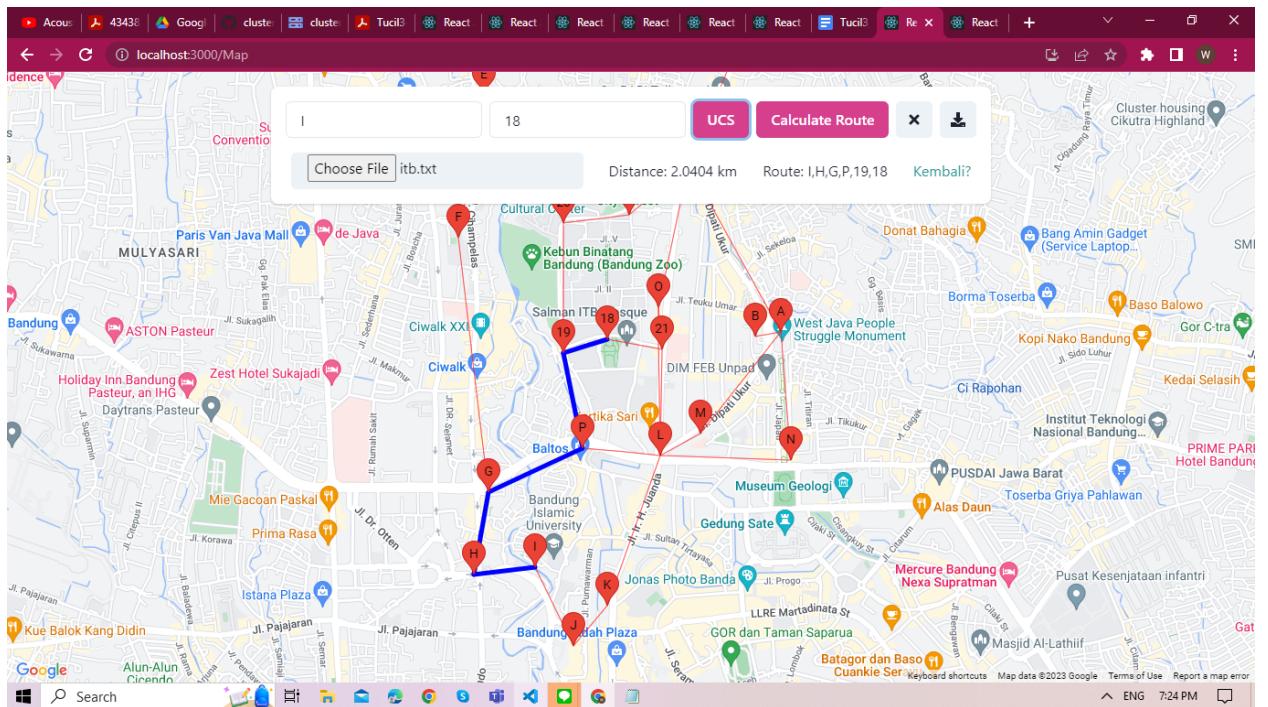


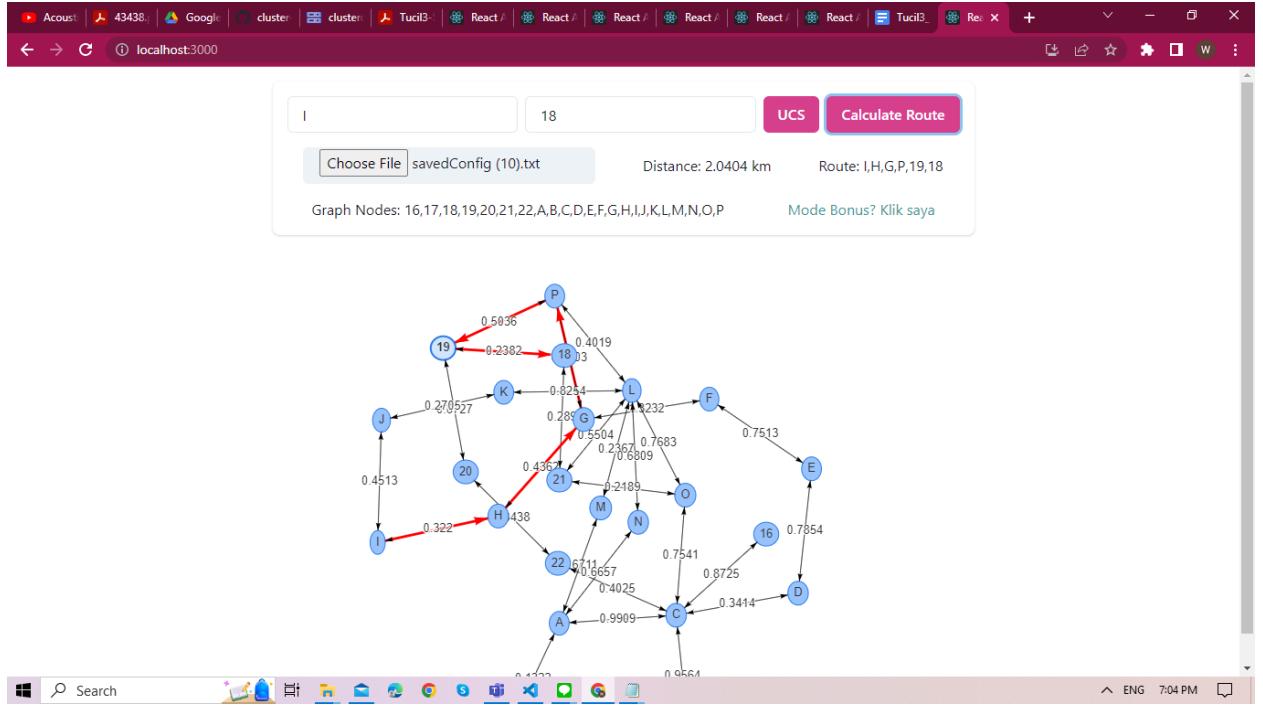
c. Output Metode A* (Node Sumber I, Node Tujuan 18)





d. Output UCS (Node Sumber I, Node Tujuan 18)

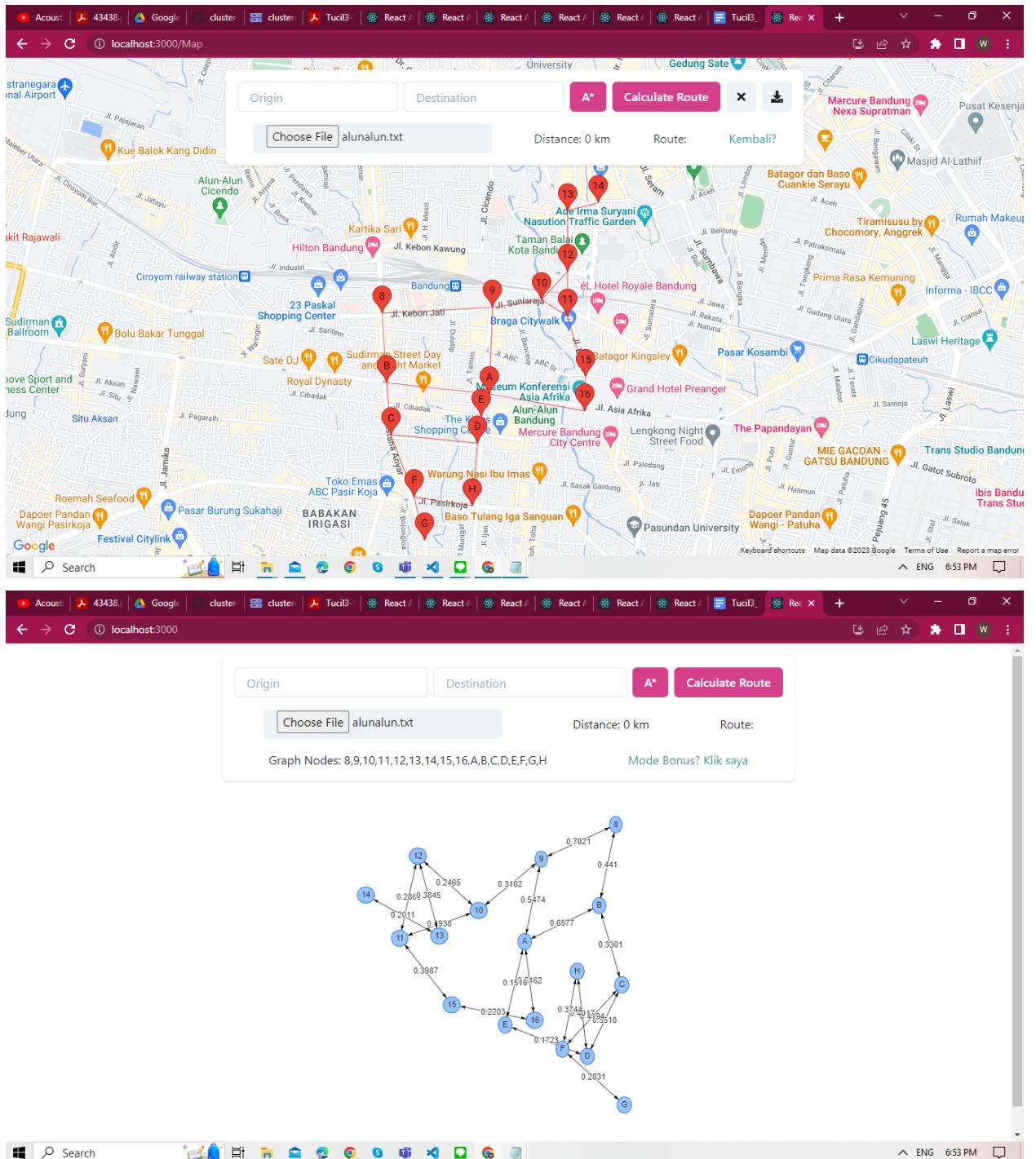




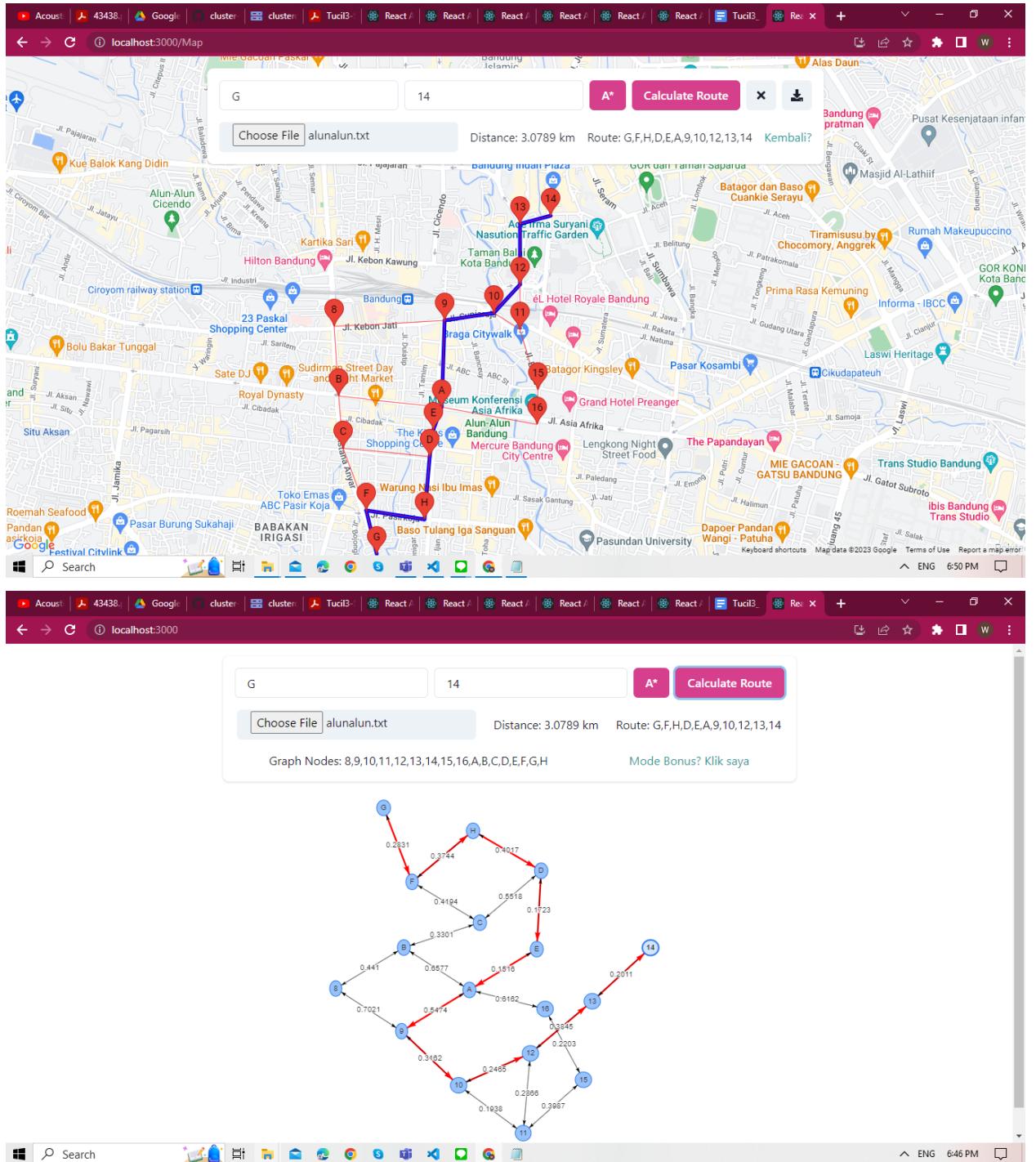
2. Peta Alun-Alun

a. Input: alunalun.txt

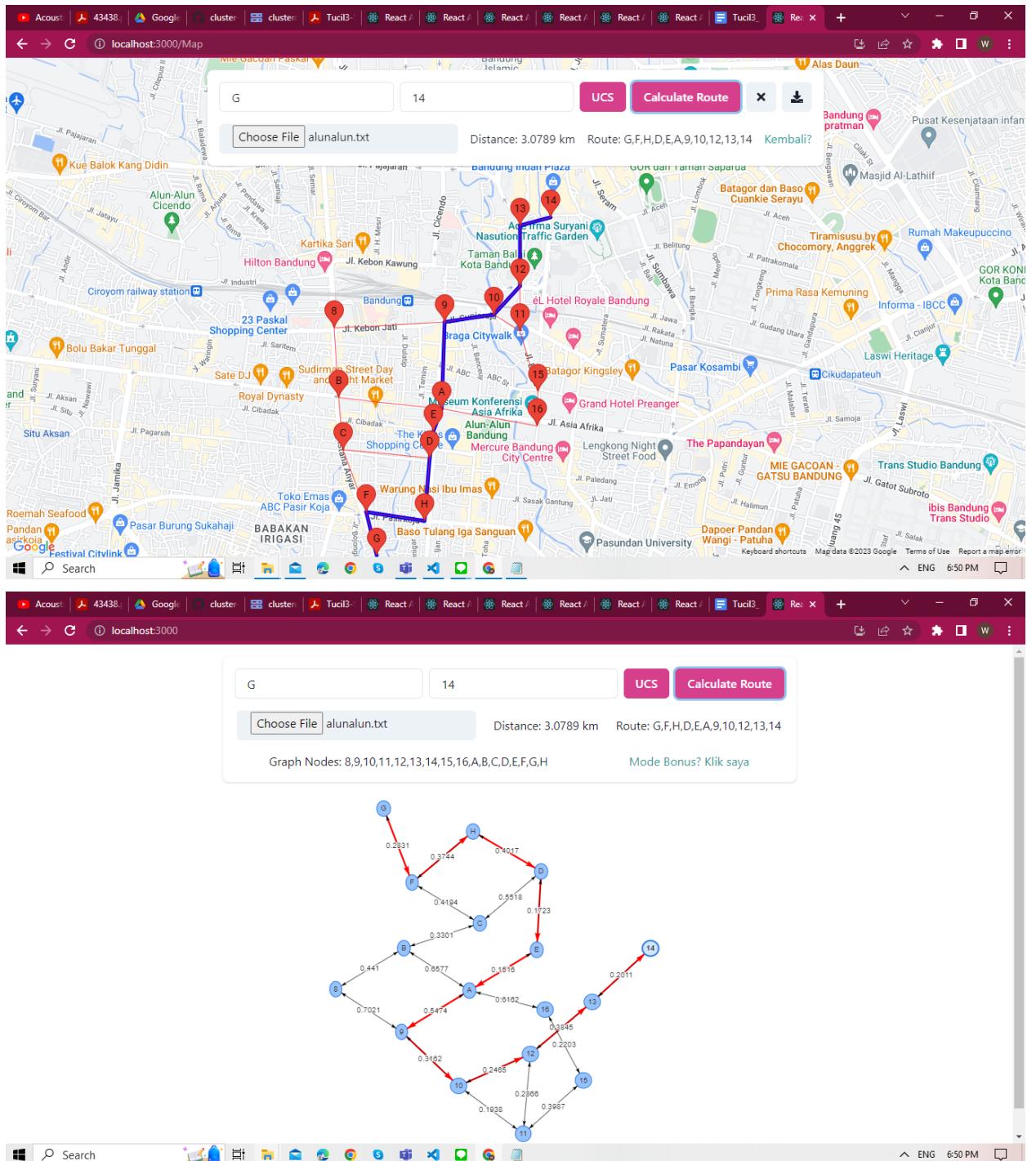
b. Tampilan Peta Inisial



c. Metode A* (Node Sumber G, Node Tujuan 14)



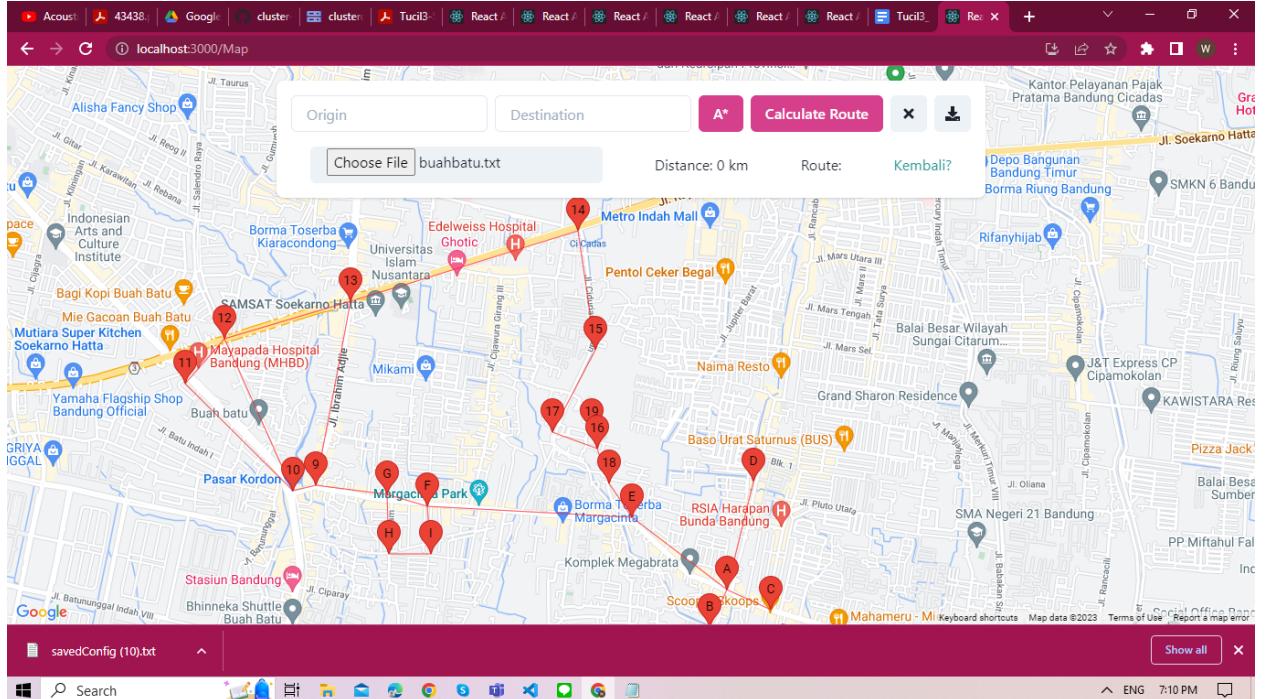
d. Metode UCS (Node Sumber G, Node Tujuan 14)

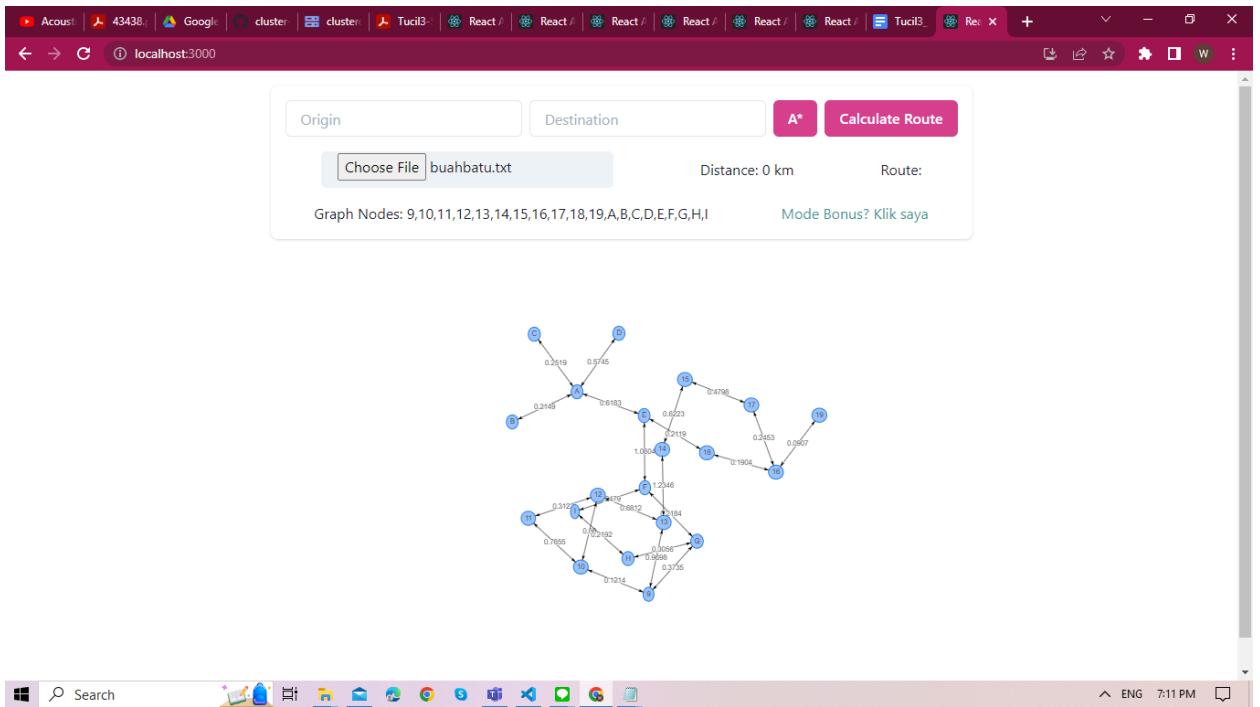


3. Peta Buah Batu

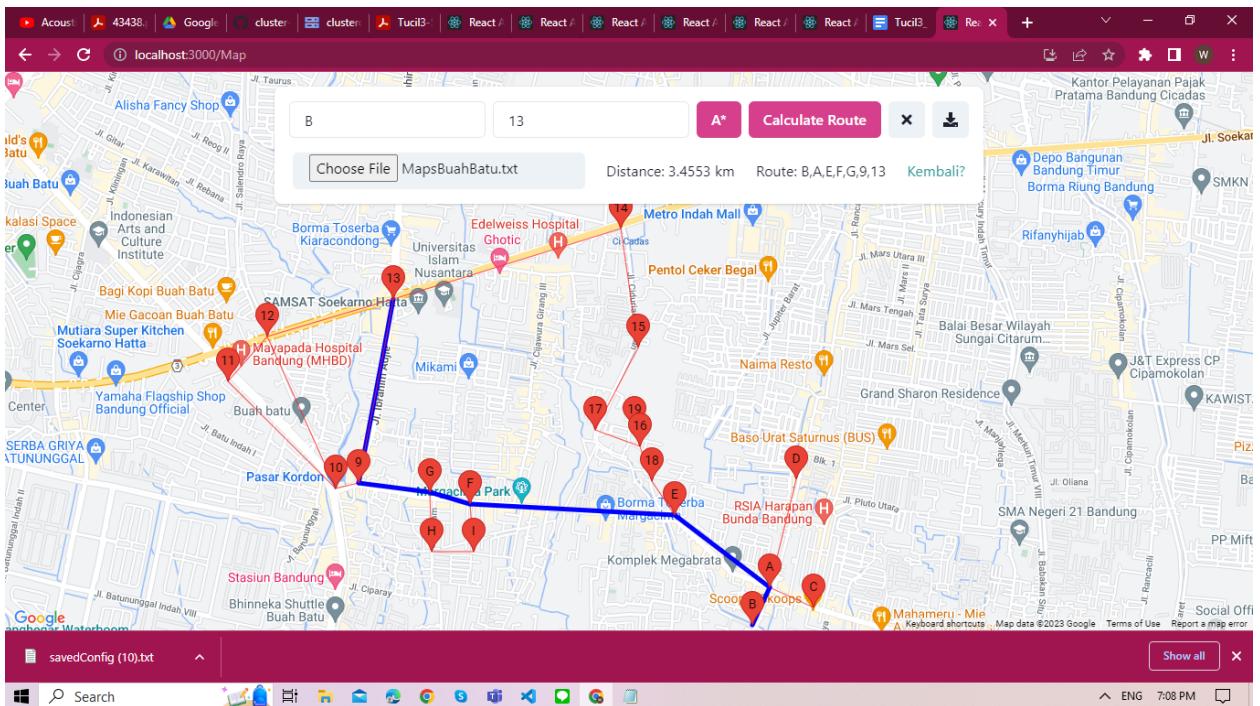
a. Input: buahbatu.txt

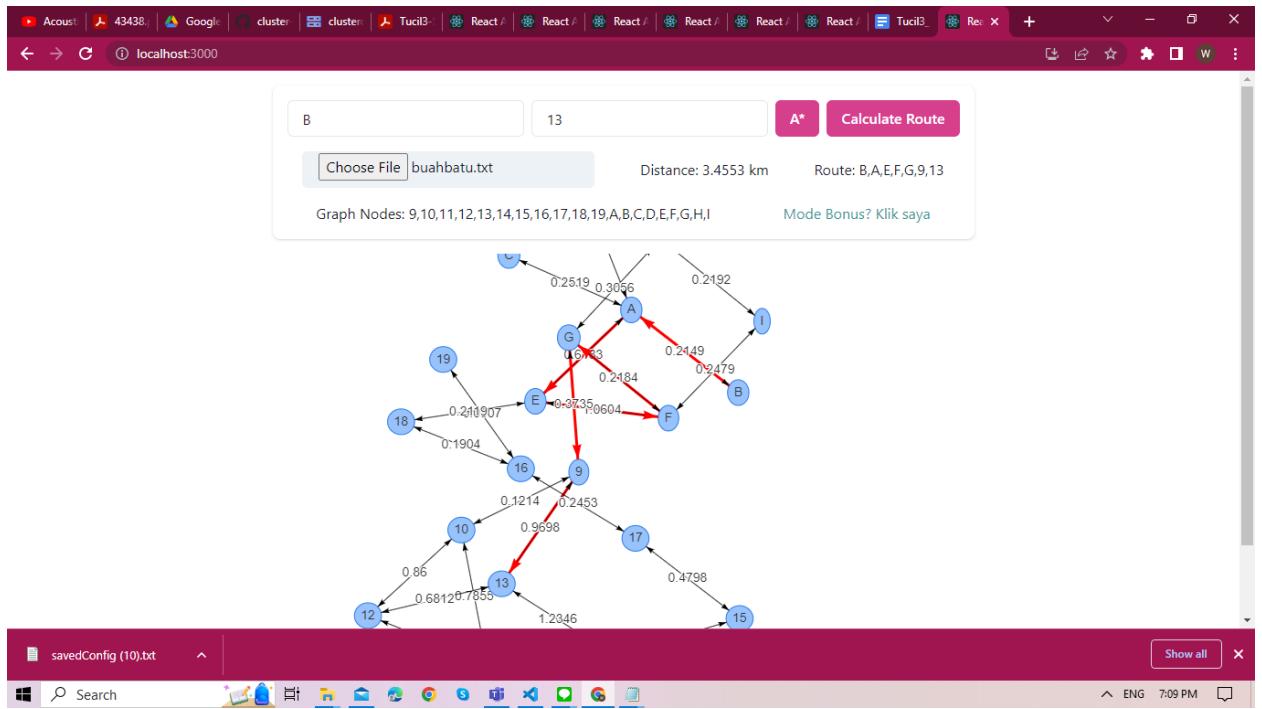
b. Tampilan Inisial



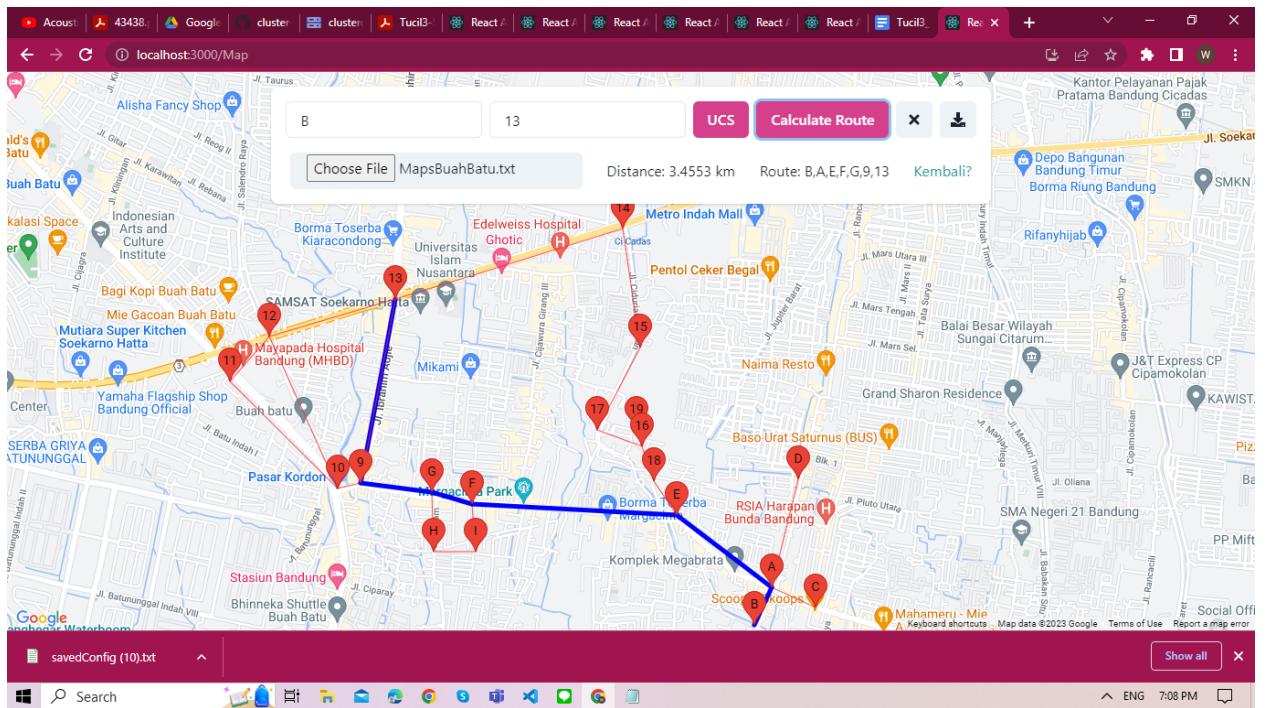


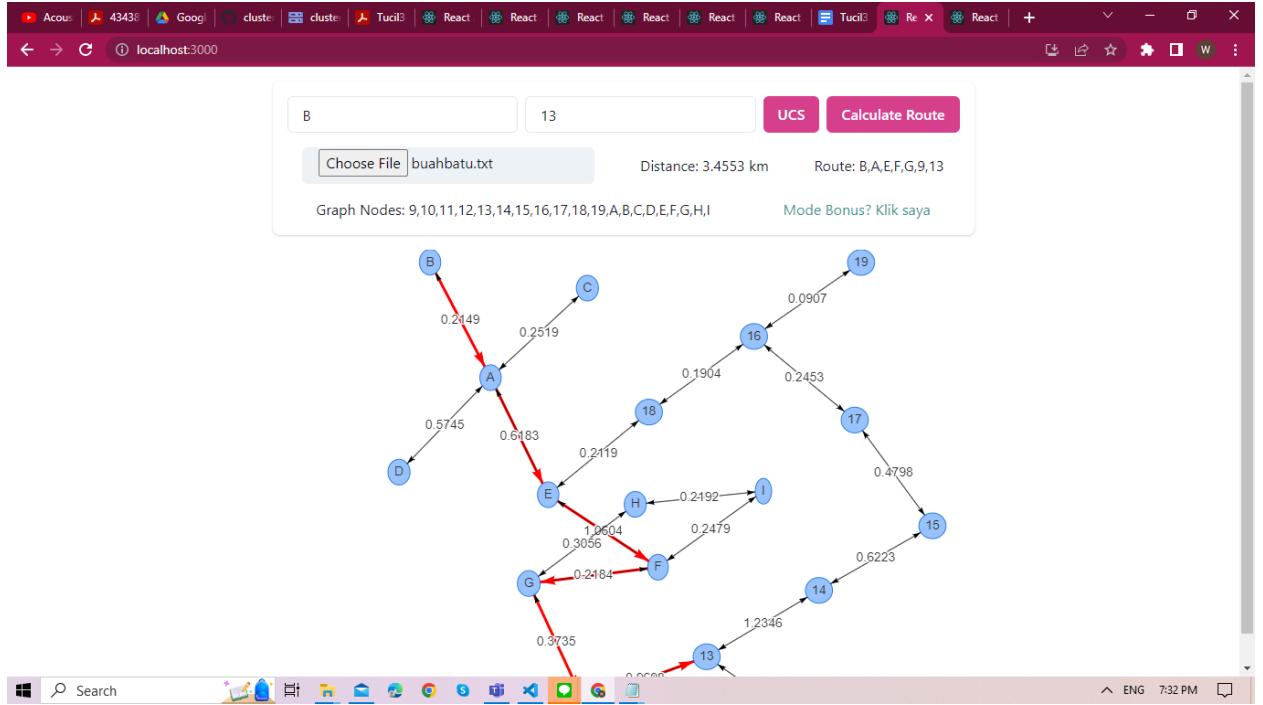
c. Metode A* (Node Sumber B, Node Tujuan 13)





d. Metode UCS (Node Sumber B, Node Tujuan 13)





4. Peta Jakarta Pusat

a. Input: jakpus.txt

b. Tampilan Inisial

localhost:3000/Map

Choose File: jakpus.txt

Distance: 0 km

Route: Kembali?

UCS Calculate Route

localhost:3000

Origin Destination

Choose File: jakpus.txt

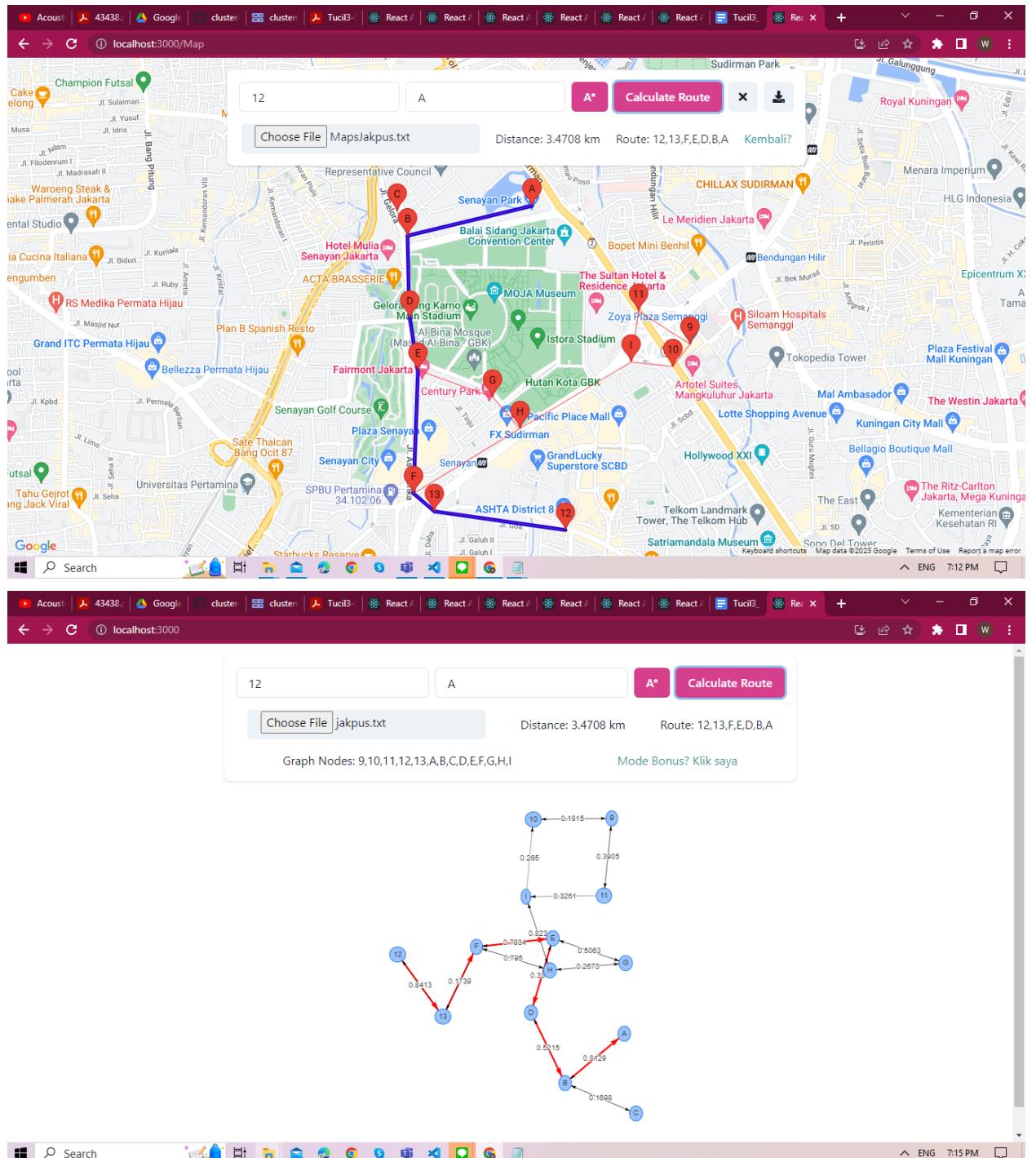
Distance: 0 km

Route:

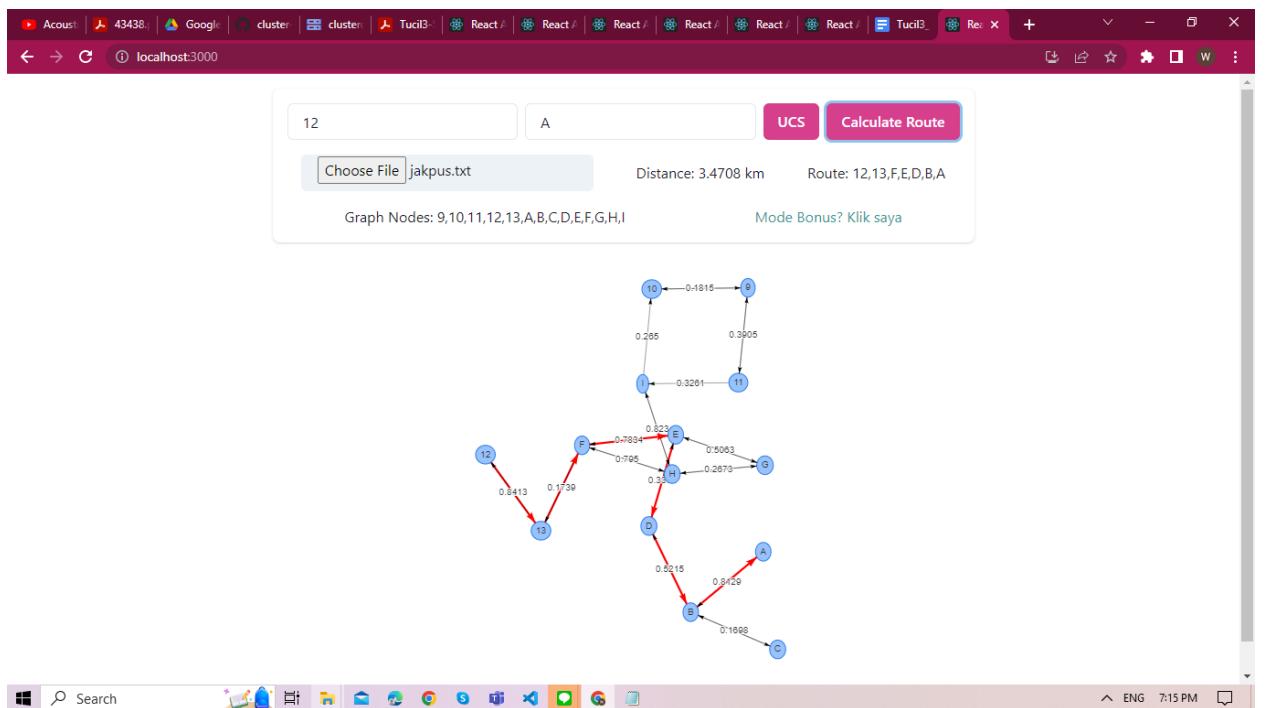
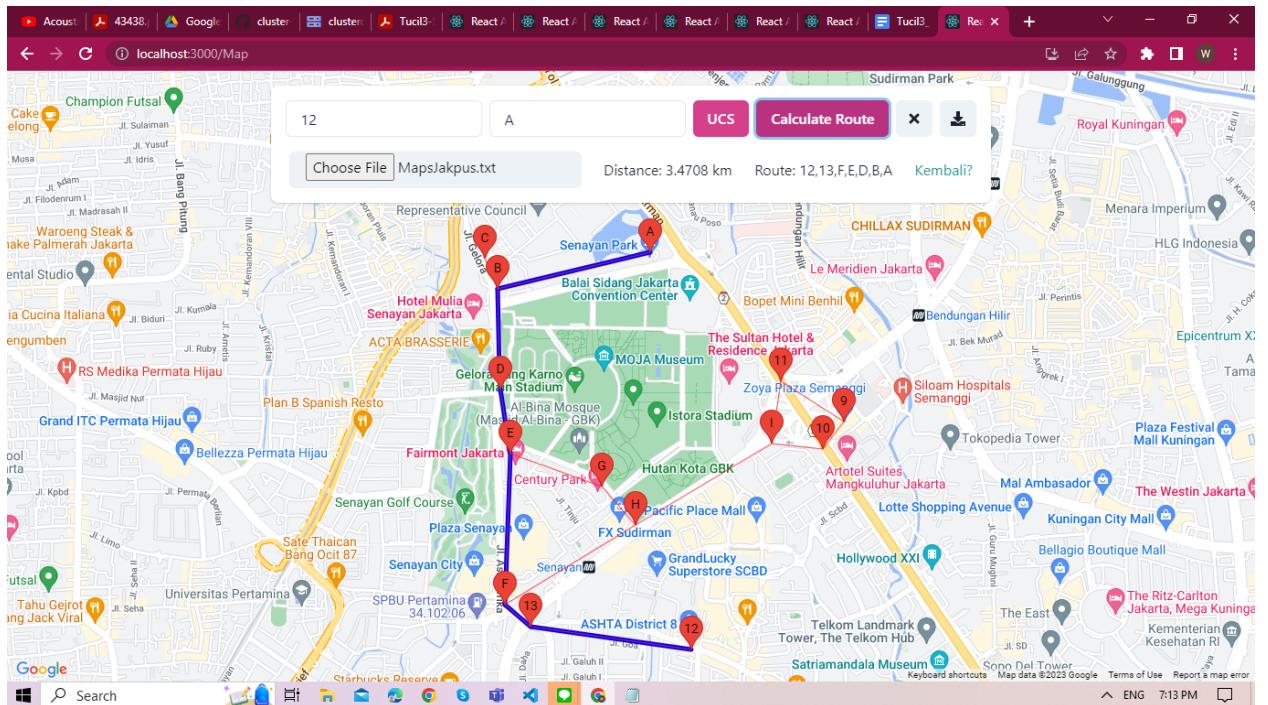
Graph Nodes: 9,10,11,12,13,A,B,C,D,E,F,G,H,I

Mode Bonus? Klik saya

C. Metode A* (Node Sumber 12, Node Tujuan A)



D. Metode UCS (Node Sumber 12- Node Tujuan A)



Bagian V

Kesimpulan dan Saran

a. Kesimpulan

Dalam tugas kecil 3 ini, kami telah mengimplementasikan program untuk menentukan lintasan terpendek antara dua simpul pada peta Google Map kota Bandung menggunakan algoritma UCS dan A*. Algoritma UCS adalah algoritma pencarian tanpa informasi yang bekerja dengan mencari simpul terendah berdasarkan jarak akumulatif dari simpul asal. Sedangkan algoritma A* adalah algoritma pencarian yang *informed* yang memanfaatkan fungsi heuristik untuk menambahkan informasi tentang jarak yang diperlukan ke tujuan. Dalam kasus ini, fungsi heuristik yang digunakan adalah jarak garis lurus dari suatu titik ke tujuan.

Hasil yang diberikan oleh kedua algoritma tersebut sama-sama merupakan lintasan terpendek antara simpul asal dan simpul tujuan. Lintasan ini dapat ditampilkan pada peta/graf dengan memberi warna merah pada jalan-jalan yang menjadi bagian dari lintasan terpendek. Dalam proses pembentukan graf, simpul merepresentasikan persilangan jalan atau ujung jalan, sedangkan bobot graf menyatakan jarak antara dua simpul yang kami hitung berdasarkan nilai *latitude* dan *longitude* simpul.

Dengan menggunakan kedua algoritma ini, kami dapat menemukan lintasan terpendek dengan lebih efisien dibandingkan dengan melakukan pencarian lintasan secara manual.

b. Saran

Terima kasih atas tugas yang sangat “menyenangkan” ini. Menurut kami, masih ada beberapa hal yang tidak jelas pada spek sehingga QNA berasa spek tambahan.

Bagian VI

Lain-Lain

1. Link Repository

<https://github.com/williamnixon20/Tucil3Stima>

2. Checklist Penilaian

Poin	Ya	Tidak
Program dapat menerima input graf	Ya	
Program dapat menghitung lintasan terpendek dengan UCS	Ya	
Program dapat menghitung lintasan terpendek dengan A*	Ya	
Program dapat menampilkan lintasan terpendek serta jaraknya	Ya	
Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	Ya	