Laporan Tugas Kecil 3 IF2122 Strategi Algoritma
Semester II Tahun 2022/2023

Implementasi Algoritma UCS dan A* Untuk Menentukan Lintasan Terpendek

Disusun oleh :

Brian Kheng (13521049)

Muhammad Habibi Husni (13521169)

PROGRAM STUDI TEKNIK INFORMATIKA

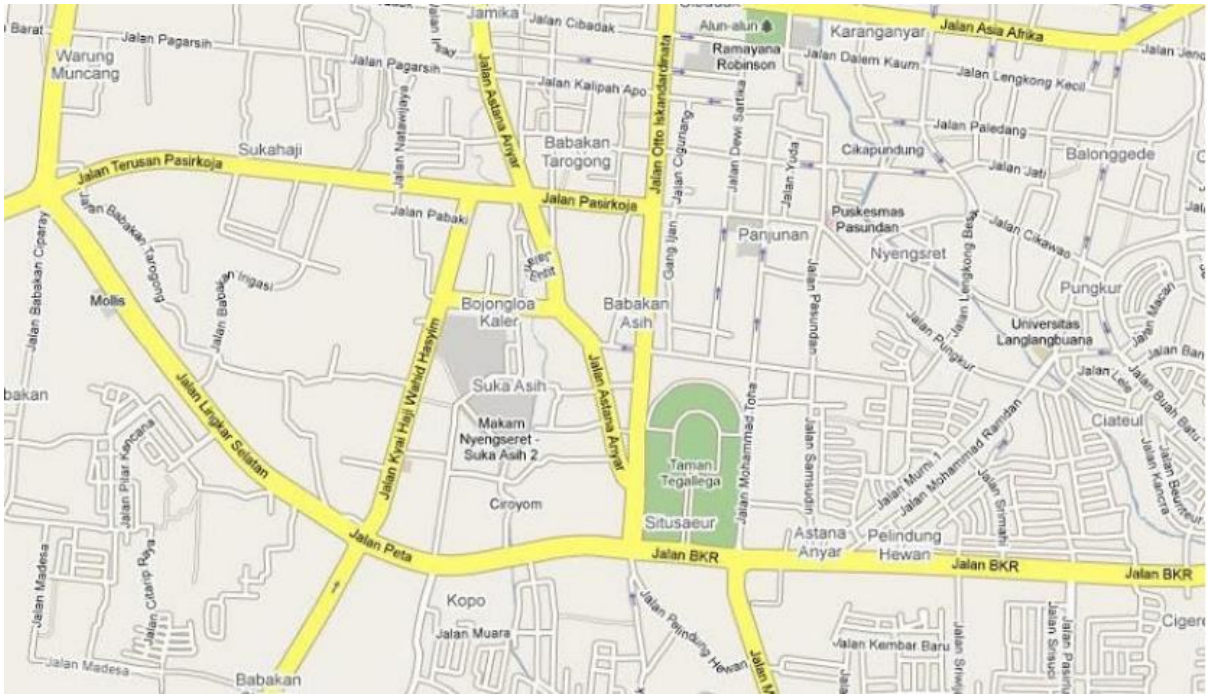
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

I. Deksripsi 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, diminta untuk membuat program yang dapat 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.

II. Kode Program

1. *inputFileHandlers.ts*

```ts
import parseData from "./parseData";

function inputFileHandler(event: any, setFileData: any, setShortestPath: any)
{
  // TODO:Handle error
  if (
    event.target.files[0] === null ||
    event.target.files[0] === undefined ||
    event.target.files[0] === "" ||
    event.target.files[0].type !== "text/plain"
  ) {
    return;
  }
  setShortestPath(null);

  const file = event.target.files[0];
  const reader = new FileReader();

  reader.readAsText(file);
  reader.onload = () => {
    setFileData(parseData(reader.result as string));
  };
}

export default inputFileHandler;
```

2. *parseData.ts*

```ts
import { IData } from "@/types";

function parseData(data: string): IData {
  const lines = data.split("\n");
  const num_nodes = parseInt(lines[0]);
  const nodes: {
    id: number;
    name: string;
    lat: number;
    lng: number;
  }[] = [];
  const adj_list: number[][] = [];
  const paths: {
    lat_start: number;
    lng_start: number;
    lat_end: number;
```

```
    lng_end: number;
  }[] = [];

  for (let i = 1; i <= num_nodes; i++) {
    const line = lines[i].split(" ");
    nodes.push({
      id: i - 1,
      name: line[2],
      lat: parseFloat(line[0]),
      lng: parseFloat(line[1]),
    });
  }

  for (let i = num_nodes + 1; i < lines.length; i++) {
    const line = lines[i].split(" ");
    const row: number[] = [];
    for (let j = 0; j < line.length; j++) {
      if (parseInt(line[j]) == 1) row.push(j);

      if (j <= i - (num_nodes + 1) && parseInt(line[j]) == 1)
        paths.push({
          lat_start: nodes[i - num_nodes - 1].lat,
          lng_start: nodes[i - num_nodes - 1].lng,
          lat_end: nodes[j].lat,
          lng_end: nodes[j].lng,
        });
    }
    adj_list.push(row);
  }
  return {
    num_nodes,
    nodes,
    adj_list,
    paths,
  };
}

export default parseData;
```

3. *UCS.ts*

```
import haversine from "./haversine";

function UCS(start: any, end: any, num_nodes: any, nodes: any, adj_list: any)
{
  const shortestPath = [];
  let shortestDistance: number = 0;
```

```
  const visited: boolean[] = Array.from({ length: num_nodes }, () => false);
  const distance: number[] = Array.from({ length: num_nodes }, () =>
Infinity);
  const parent: number[] = Array.from({ length: num_nodes }, () => -1);
  const priorityQueue: [number, number][] = [];

  distance[start] = 0;
  priorityQueue.push([start, 0]);

  while (priorityQueue.length > 0) {
    priorityQueue.sort((a, b) => a[1] - b[1]);
    const [idx, _] = priorityQueue.shift()!;
    if (visited[idx]) continue;
    visited[idx] = true;

    for (const it of adj_list[idx]) {
      const newDistance = distance[idx] + haversine(nodes[idx], nodes[it]);
      if (newDistance < distance[it]) {
        distance[it] = newDistance;
        parent[it] = idx;
        priorityQueue.push([it, newDistance]);
      }
    }
  }

  // Check if there is no path
  if (distance[end] === Infinity) throw new Error("No path found!");

  let idx = end;
  while (idx != start) {
    shortestPath.push({
      id: shortestPath.length,
      lat_start: nodes[parent[idx]].lat,
      lng_start: nodes[parent[idx]].lng,
      lat_end: nodes[idx].lat,
      lng_end: nodes[idx].lng,
    });
    shortestDistance += haversine(nodes[parent[idx]], nodes[idx]);
    idx = parent[idx];
  }

  shortestDistance = Math.round(shortestDistance * 1000) / 1000000;

  return { shortestPath, shortestDistance };
}

export default UCS;
```

4. *Astar.ts*

```
import haversine from "./haversine";

function AStar(start: any, end: any, num_nodes: any, nodes:any, adj_list: any)
{
  const shortestPath = [];
  let shortestDistance: number = 0;
  const visited: boolean[] = Array.from({length : num_nodes}, () => false);
  const distance: number[] = Array.from({length : num_nodes}, () => Infinity);
  const parent: number[] = Array.from({ length: num_nodes }, () => -1);
  const priorityQueue: [number, number][] = [];
  const heuristic: Map<number, number> = new Map<number, number>();

  for (const node of nodes) {
    heuristic.set(node.id, haversine(node, nodes[end]));
  }

  distance[start] = 0;
  priorityQueue.push([start, 0+heuristic.get(start)!]);
  while (priorityQueue.length > 0) {
    priorityQueue.sort((a, b) => a[1] - b[1]);
    const [now, _] = priorityQueue.shift()!;
    if (visited[now]) continue;
    if (now == end) break;
    visited[now] = true;

    for (const next of adj_list[now]) {
      const newDistance = distance[now] + haversine(nodes[now], nodes[next]);
      if (newDistance < distance[next]) {
        distance[next] = newDistance;
        parent[next] = now;
        priorityQueue.push([next, newDistance+heuristic.get(next)!]);
      }
    }
  }

  // Check if there is no path
  if(distance[end] === Infinity) throw new Error("No path found!");

  let idx = end;
  while (idx != start) {
    shortestPath.push({
      id: shortestPath.length,
      lat_start: nodes[parent[idx]].lat,
      lng_start: nodes[parent[idx]].lng,
      lat_end: nodes[idx].lat,
      lng_end: nodes[idx].lng,
    });
```

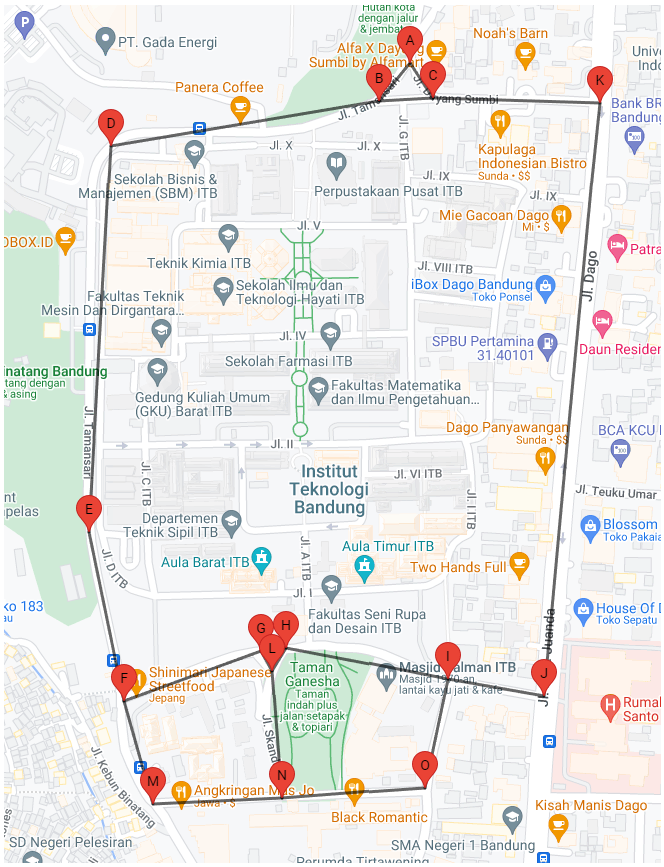
```
    shortestDistance += haversine(nodes[parent[idx]], nodes[idx]);
    idx = parent[idx];
  }
  shortestDistance = Math.round(shortestDistance * 1000) / 1000000;
  return { shortestPath, shortestDistance };
}

export default AStar;
```

5. *haversine.ts*

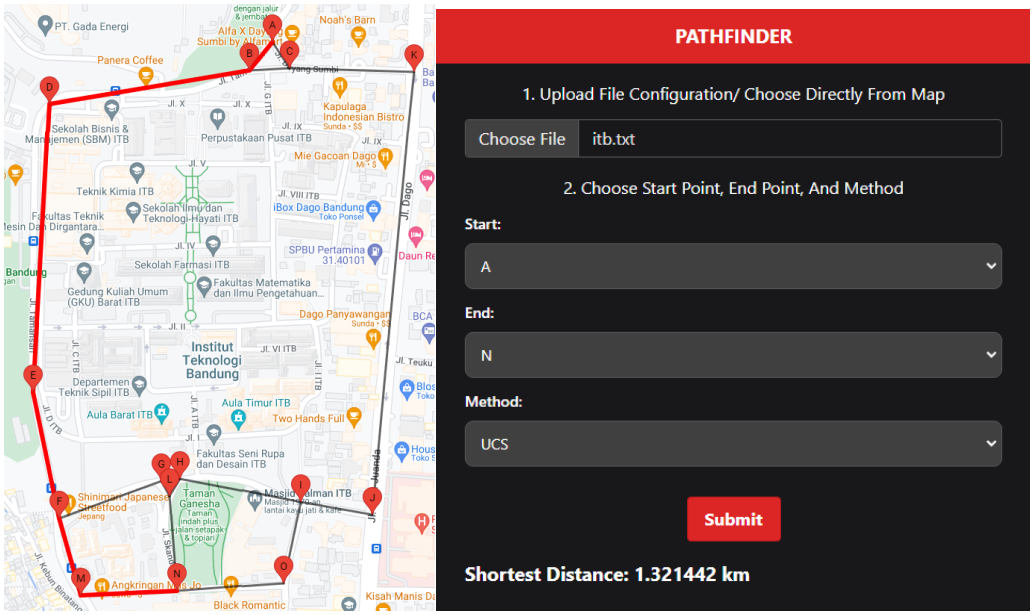
```
function haversine(start: any, end: any) {
  const R = 6371e3;
  const φ1 = (start.lat * Math.PI) / 180;
  const φ2 = (end.lat * Math.PI) / 180;
  const Δφ = ((end.lat - start.lat) * Math.PI) / 180;
  const Δλ = ((end.lng - start.lng) * Math.PI) / 180;

  const a =
    Math.sin(Δφ / 2) * Math.sin(Δφ / 2) +
    Math.cos(φ1) * Math.cos(φ2) * Math.sin(Δλ / 2) * Math.sin(Δλ / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

  const d = R * c;
  return d;
}

export default haversine;
```

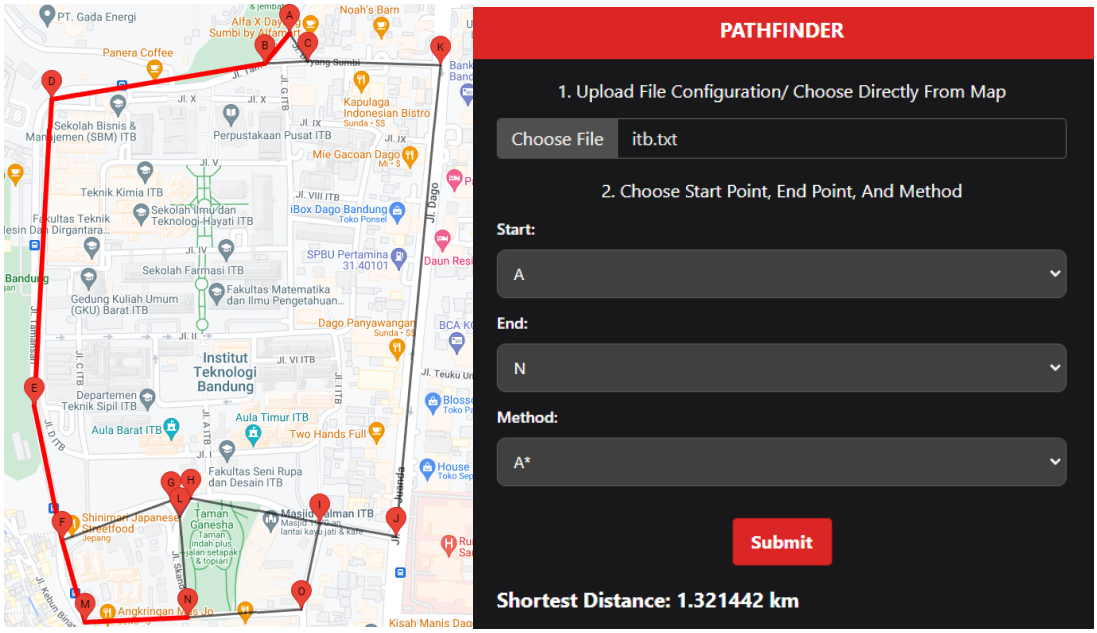
III. Uji Coba

1. Peta Daerah Sekitar ITB

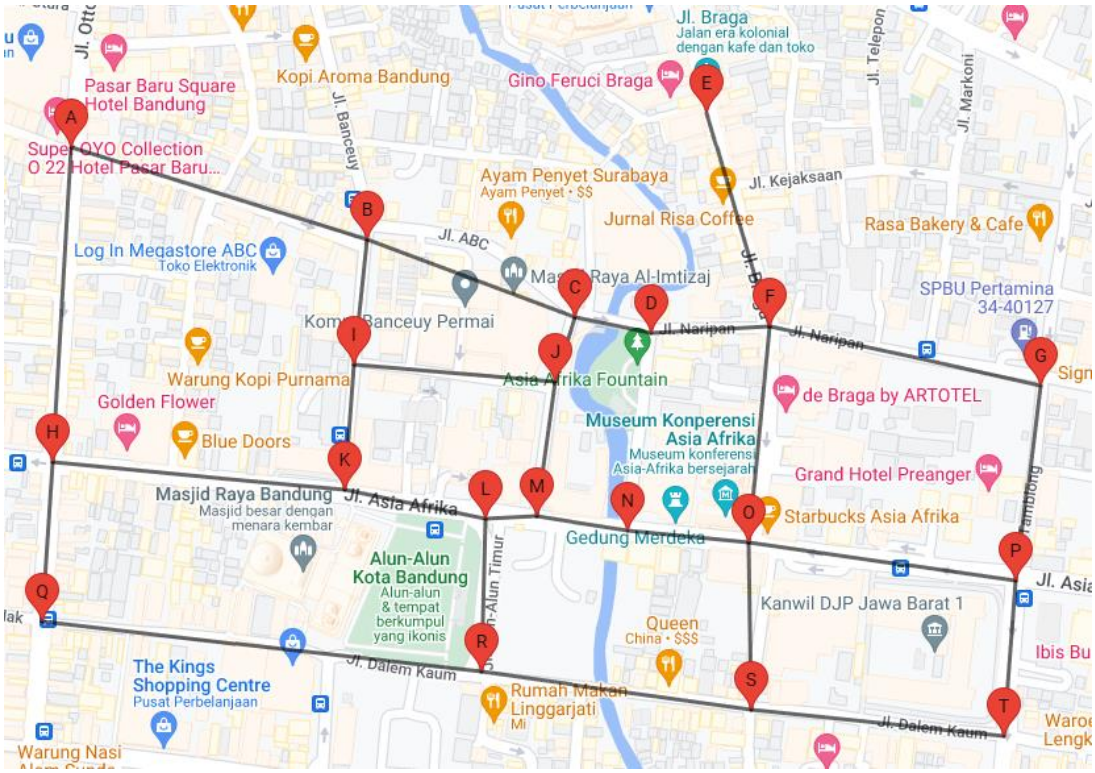


Hasil:

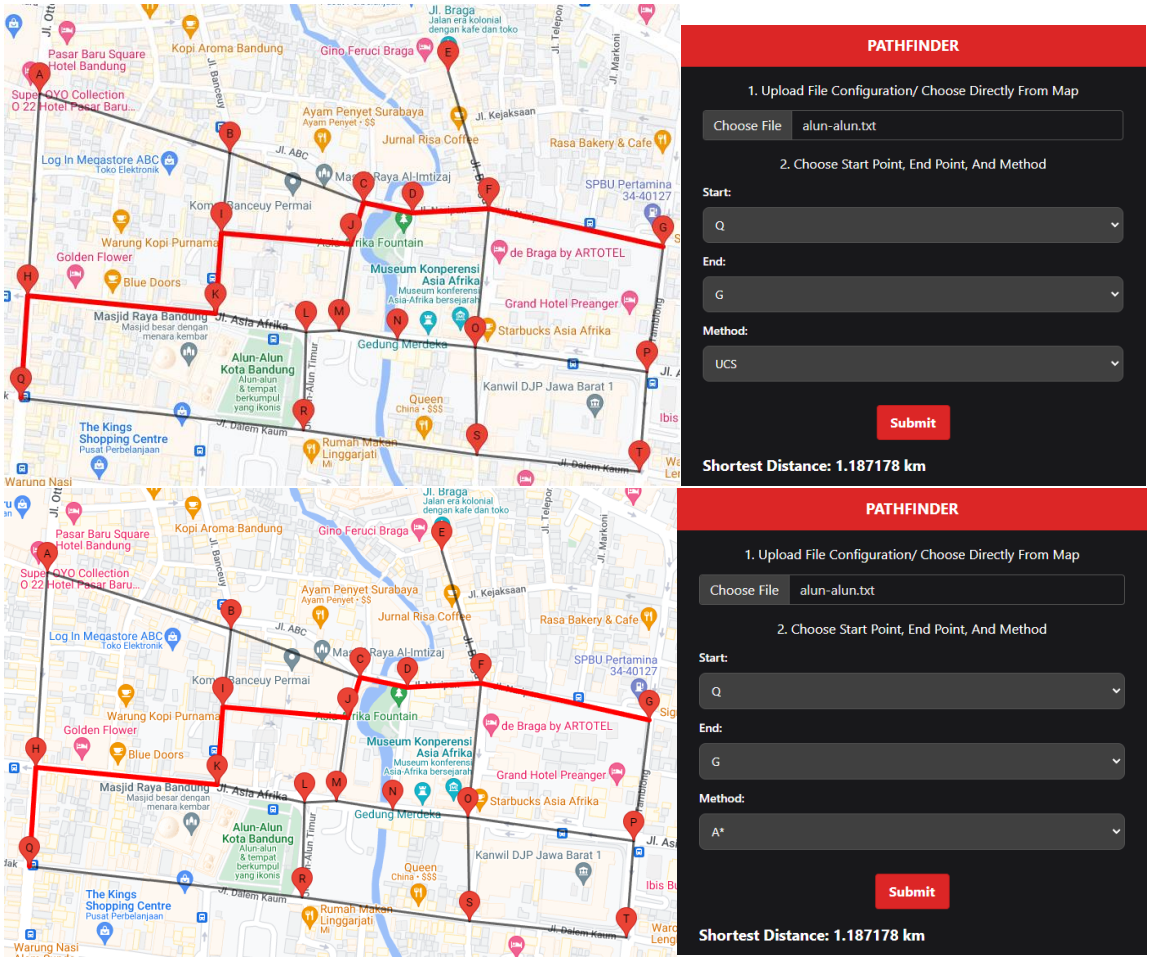




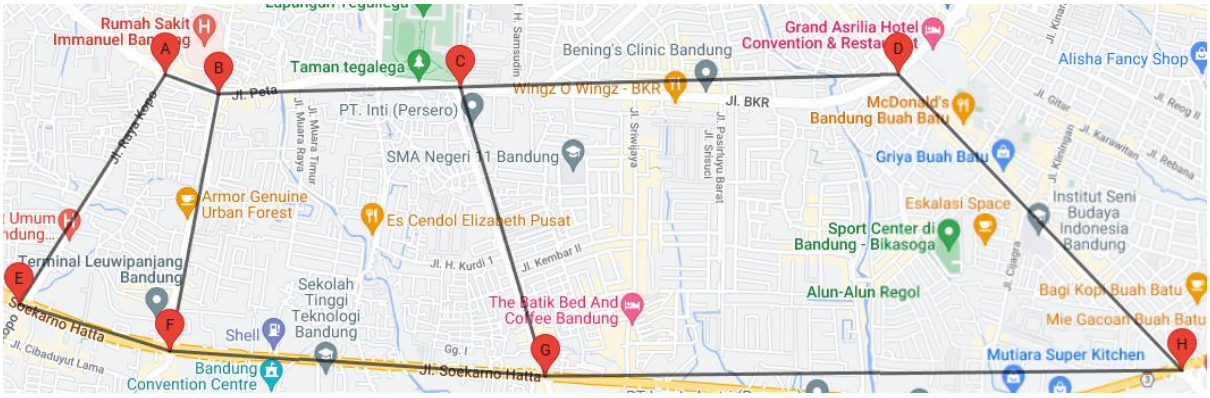
2. Peta Daerah Sekitar Alun-Alun



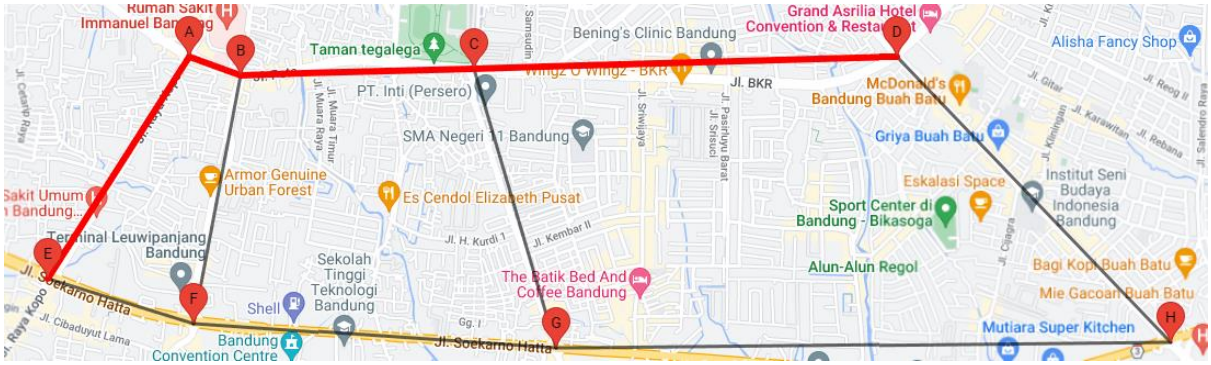
Hasil:



3. Peta Daerah Bandung Selatan



Hasil:



PATHFINDER

1. Upload File Configuration/ Choose Directly From Map

Choose File | bandung-selatan.txt

2. Choose Start Point, End Point, And Method

Start:

E

End:

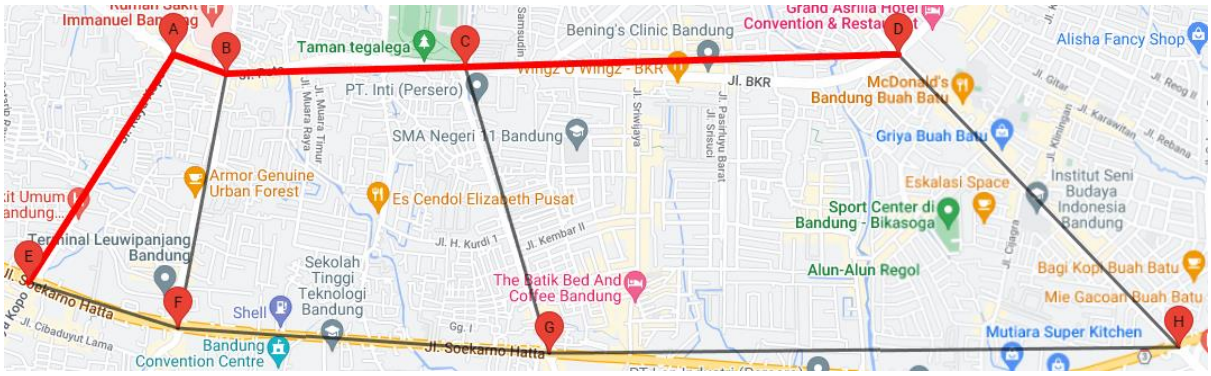
D

Method:

UCS

Submit

Shortest Distance: 4.207987 km



PATHFINDER

1. Upload File Configuration/ Choose Directly From Map

Choose File | bandung-selatan.txt

2. Choose Start Point, End Point, And Method

Start:

E

End:

D

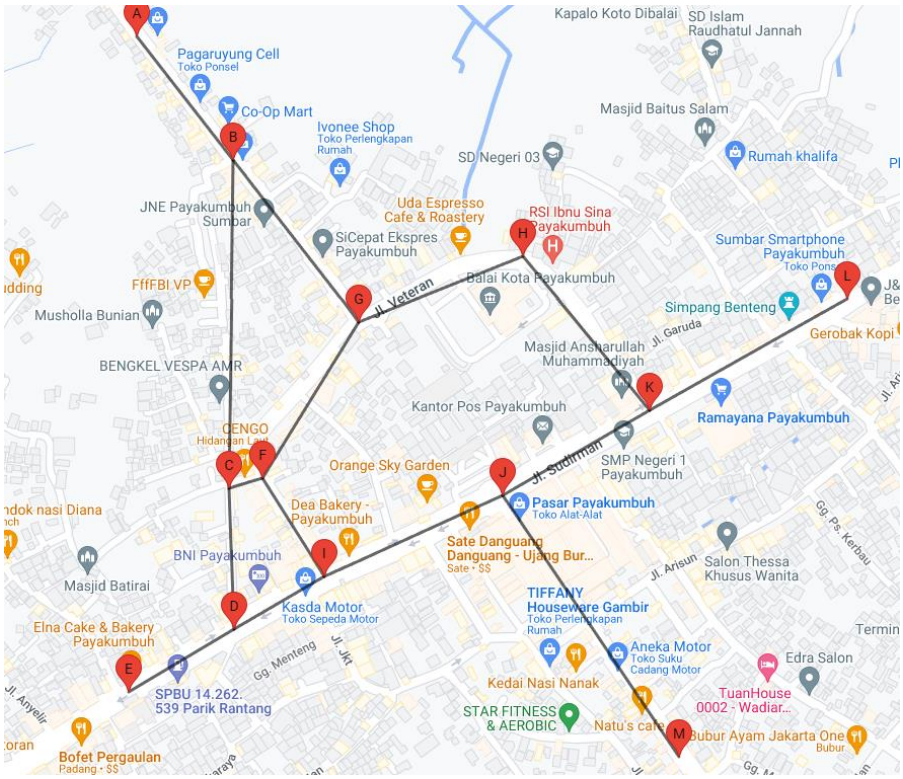
Method:

A*

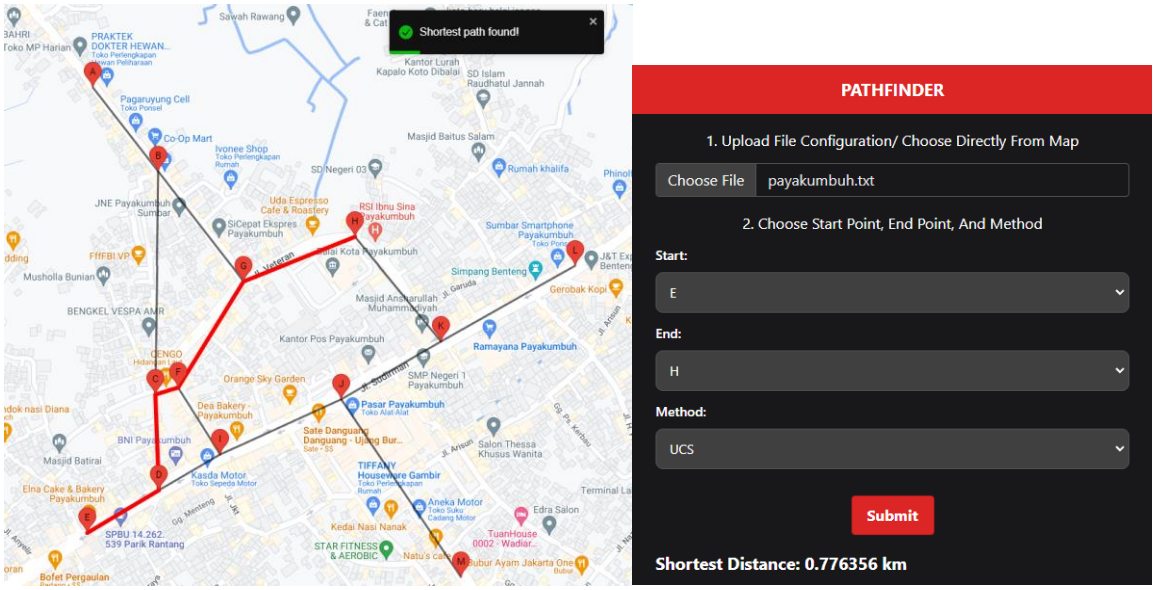
Submit

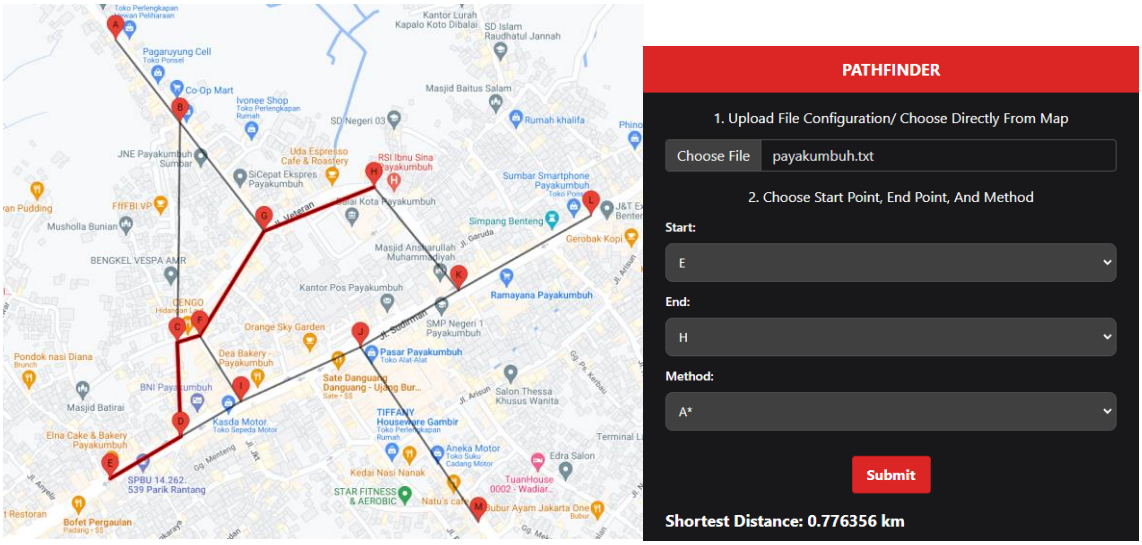
Shortest Distance: 4.207987 km

4. Peta Daerah Payakumbuh, Sumatera Barat



Hasil:





IV. Kesimpulan

Pada tugas ini, dapat dilihat bahwa untuk kedua algoritma dapat mengembalikan rute paling optimal dari satu simpul ke satu simpul lainnya. Pada algoritma UCS, hanya digunakan jarak yang telah ditempuh pada pencarian untuk menentukan rute terpendek. Pada algoritma A* digunakan jarak yang ditempuh serta pendekatan fungsi heuristik berupa jarak *haversine* antara dua koordinat simpul. Dikarenakan fungsi yang dipilih bersifat *admissible*, maka algoritma A* dipastikan mengembalikan rute yang paling optimal sesuai dengan algoritma UCS.

V. Lampiran

a) Tabel Evaluasi

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	✅

b) Pranala Github
https://github.com/briankheng/Tucil3_13521049_13521169

c) Pranala Website
https://pathfinder-gold.vercel.app/