Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Laporan Tugas Kecil

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma

Semester II 2020/2021

Disusun oleh

Reyhan Emyr Arrosyid (13519167)

Muhammad Dehan Al Kautsar (13519200)

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2021

1. Source Code

Nama File : Vertex.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Tucil3.Graph
{
    class Vertex
    {
        // Nama dari simpul
        public string Name
        {
            get;
            set;
        }

        // Simpul yang bersisian
        public List<Edge> Edges
        {
            get;
            set;
        }

        // Koordinat latitude
        public double Latitude
        {
            get;
            set;
        }

        // Koordinat longitude
        public double Longitude
        {
            get;
            set;
        }

        // Constructor
        public Vertex(string name, double lat, double lon)
        {
            this.Name = name;
            this.Edges = new List<Edge>();
            this.Latitude = lat;
            this.Longitude = lon;
        }
    }
}
```

Nama File : Edge.cs

```cs
using System;
using System.Collections.Generic;
using System.Text;

namespace Tucil3.Graph
{
    class Edge
    {
        // Nama simpul tujaun
        public string ToVertex
        {
            get;
            set;
        }

        // Weight sisi
        public double Weight
        {
            get;
            set;
        }

        // Constructor
        public Edge(string v, double w)
        {
            ToVertex = v;
            Weight = w;
        }
    }
}
```

Nama File = Graph.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
using System.IO;
using System.Globalization;

namespace Tucil3.Graph
{
    class Graph
    {
        // Simpul dalam graph
        public List<Vertex> Vertices;

        // Default constructor
        public Graph()
        {
            Vertices = new List<Vertex>();
        }

        // Constructor menerima nama file
        public Graph(string filename)
        {
            Vertices = new List<Vertex>();
            string[] filePerLine = File.ReadAllLines(filename); //baca file per line
            //hitung jumlah vertex
            int nVertex = int.Parse(filePerLine[0]);
            //container isi file per line
            string[] pairVertex;
            string[] pairEdge;
            //buat nyari nama vertex pas masukin edge adj matrix
            Dictionary<int, string> kamusVertex = new Dictionary<int, string>();

            int i = 1;
            for (; i <= nVertex; i++)
            {
                pairVertex = filePerLine[i].Split(' '); //baca vertex

                //karena urutan di file : lintang bujur nama
                this.AddVertex(
                    pairVertex[2],
                    double.Parse(pairVertex[0], CultureInfo.InvariantCulture),
                    double.Parse(pairVertex[1], CultureInfo.InvariantCulture));
                kamusVertex.Add(i - 1, pairVertex[2]); //nambahin nama vertex ke kamus
            }

            for (int j = i; j < filePerLine.Length; j++)
            {
                pairEdge = filePerLine[j].Split(' '); //baca adj matrix
                for (int k = 0; k <= j-i; k++)
                {
                    if (double.Parse(pairEdge[k]) > 0)
                    {
                        //intinya ini masukin edge berasal dari kamus nama
                        this.AddEdge(
                            kamusVertex[j-i],
                            kamusVertex[k],
                            double.Parse(pairEdge[k],
                            CultureInfo.InvariantCulture));
                    }
                }
            }
        }
```

```
        // Menambah simpul
        public void AddVertex(string v, double x, double y)
        {
            Vertices.Add(new Vertex(v, x, y));
        }

        // Menambah sisi (Asumsi nama simpul valid)
        public void AddEdge(string v1Name, string v2Name, double w)
        {
            Edge e1 = new Edge(v2Name, w);
            Edge e2 = new Edge(v1Name, w);
            Vertex v1 = Vertices.Find(v => v.Name == v1Name);
            Vertex v2 = Vertices.Find(v => v.Name == v2Name);

           // Jika sisi sudah ada tidak ditambahkan
           foreach (Edge e in v1.Edges)
            {
                if (e.ToVertex == v2Name)
                {
                    return;
                }
            }

            v1.Edges.Add(e1);
            v2.Edges.Add(e2);
        }

        // Fungsi A Star mencari jalan dari simpul source ke simpul dest
        public Tuple<List<string>, string> AStar(string source, string dest)
        {
            // Inisiasi
            Dictionary<string, double> cost = new Dictionary<string, double>();
            Dictionary<string, string> prev = new Dictionary<string, string>();
            List<Tuple<string, double>> pq = new List<Tuple<string, double>>();

            foreach (Vertex v in Vertices)
            {
                prev[v.Name] = null;
            }

            // Cost node asal = 0
            cost[source] = 0;

            // Tambah node asal ke dalam list simpul yang akan diekspan
            pq.Add(new Tuple<string, double>(source, 0));

            // Selama masih ada simpul yang dapat diekspan
            while (pq.Count > 0)
            {
                // Ambil simpul yang memiliki estimasi cost terendah
                double min = pq.Min(i => i.Item2);
                Tuple<string, double> current = pq.FirstOrDefault(i => i.Item2 == min);

                // Keluarkan simpul terendah
                pq.Remove(current);

                Vertex curVertex = Vertices.Find(v => v.Name == current.Item1);

                // Jika simpul saat ini sudah sama dengan tujuan, pencarian selesai
                if (curVertex.Name == dest)
                {
                    break;
                }
```

```
            // Untuk setiap sisi pada simpul saat ini
            foreach (Edge e in curVertex.Edges)
            {
                // Hitung cost baru untuk simpul tujuan sisi
                double newCost = cost[curVertex.Name] + e.Weight;

                // Jika simpul belum ada pada dictionary cost
                // atau cost baru lebih kecil dari yang lama
                if (!cost.ContainsKey(e.ToVertex) || newCost < cost[e.ToVertex])
                {
                    // Update cost simpul
                    cost[e.ToVertex] = newCost;

                    // Hitung estimasi cost menuju goal
                    double estCost = newCost + CalcHeuristic(e.ToVertex, dest);

                    // Tambahkan dalam list simpul ekspan
                    pq.Add(new Tuple<string, double>(e.ToVertex, estCost));

                    // Update nilai prev untuk simpul
                    prev[e.ToVertex] = curVertex.Name;
                }
            }
        }

        // Ambil jalur yang ditemukan
        List<string> path = GetPath(source, dest, prev);

        // String yang akan ditampilkan pada GUI
        string resultString = "";

        // Jika tidak ada jalur
        if (path == null)
        {
            resultString += "Tidak ada jalur";
        }
        else
        {
            foreach (string node in path)
            {
                resultString += node;
                if (node != path[path.Count - 1])
                {
                    resultString += " -> ";
                }
            }
            resultString += "\n";
            resultString += "Jarak yang ditempuh: ";
            if (cost[dest] >= 1000)
            {
                double km = cost[dest] / 1000;
                resultString += km.ToString() + " kilometer";
            }
            else
            {
                resultString += cost[dest].ToString() + " meter";
            }
        }

        return new Tuple<List<string>, string>(path, resultString);
    }
```

```
        // Fungsi untuk menghasilkan jalur dari source ke dest sesuai dengan prev
        public List<string> GetPath(string source, string dest, Dictionary<string, string> prev)
        {
            List<string> path = new List<string>();
            string now = dest;
            while (now != null)
            {
                path.Add(now);
                now = prev[now];
            }

            path.Reverse();

            if (path[0] == source)
            {
                return path;
            }
            else
            {
                return null;
            }
        }

        // Fungsi mengubah derajat ke radian
        public double degreeToRadian (double degree) {
            return degree*Math.PI/180;
        }

        // Fungsi haversine untuk menghitung jarak antara dua koordinat
        public double haversine(
            double curLintang,
            double goalLintang,
            double curBujur,
            double goalBujur)
        { //semua parameter belum diubah ke radian
            curBujur = degreeToRadian(curBujur);
            goalBujur = degreeToRadian(goalBujur);
            curLintang = degreeToRadian(curLintang);
            goalLintang = degreeToRadian(goalLintang);

            double diffBujur = goalBujur - curBujur;
            double diffLintang = goalLintang - curLintang;
            double haversineFormula =
                Math.Pow(
                    Math.Sin(
                        diffLintang/2),2) +
                        Math.Cos(curLintang) *
                        Math.Cos(goalLintang) *
                        Math.Pow(Math.Sin(diffBujur/2),
                    2);
            double distancePerKilometer = 2 * Math.Asin(Math.Sqrt(haversineFormula));
            return distancePerKilometer * 6378137; //6378137 itu jari jari bumi (satuan meter)
        }

        // Fungsi heuristic dua simpul
        private double CalcHeuristic(string v1, string goal)
        {
            Vertex curVertex = Vertices.Find(v => v.Name == v1);
            Vertex goalVertex = Vertices.Find(v => v.Name == goal);

            return haversine(
                curVertex.Latitude,
                goalVertex.Latitude,
                curVertex.Longitude,
                goalVertex.Longitude);
        }
```

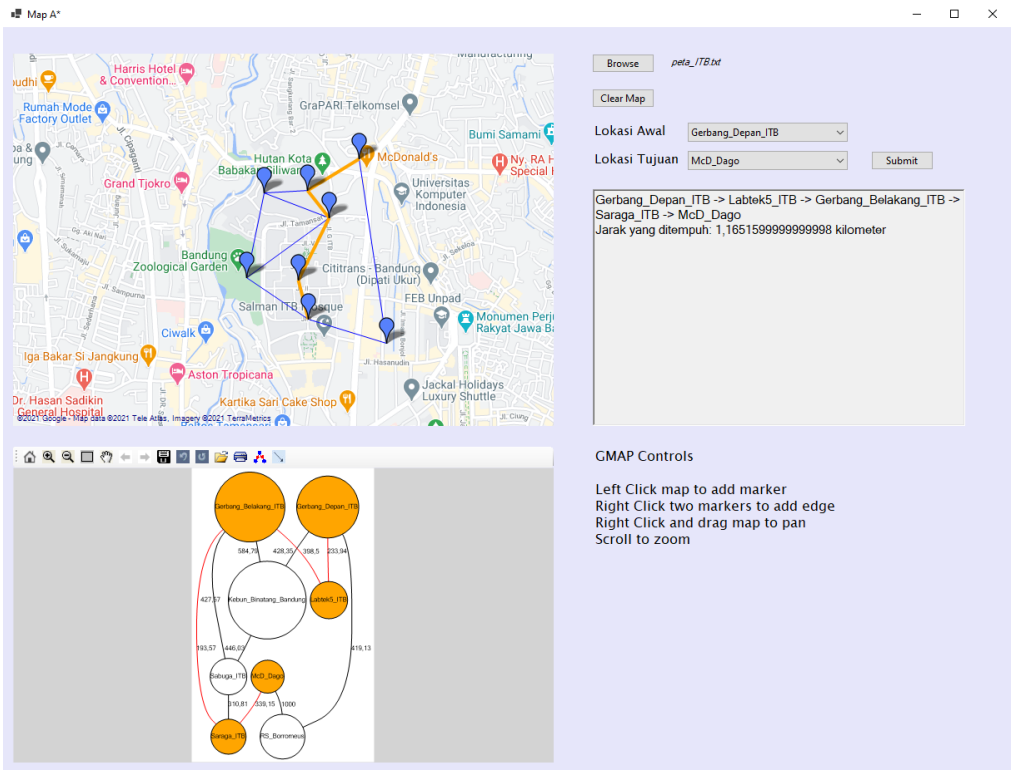
```
        // Mengembalikan graf dalam bentuk graph MSAGL untuk divisualisasikan
        public Microsoft.Msagl.Drawing.Graph getMSAGLGraph(string name)
        {
            Microsoft.Msagl.Drawing.Graph g = new Microsoft.Msagl.Drawing.Graph(name);
            g.Directed = false;
            Vertices.ForEach(
                v => g.AddNode(v.Name).Attr.Shape = Microsoft.Msagl.Drawing.Shape.Circle);

            foreach (Vertex v in Vertices)
            {
                foreach (Edge e in v.Edges)
                {
                    int a = v.Name.CompareTo(e.ToVertex);
                    if (v.Name.CompareTo(e.ToVertex) < 0)
                    {
                        Microsoft.Msagl.Drawing.Edge edge =
                            g.AddEdge(v.Name, e.Weight.ToString(), e.ToVertex);
                        edge.Attr.ArrowheadAtTarget = Microsoft.Msagl.Drawing.ArrowStyle.None;
                        edge.Attr.Id = v.Name + e.ToVertex;
                    }
                }
            }
            return g;
        }
    }
}
```

2. Peta/Graf Input yang Digunakan Beserta Hasil Tangkapan Layar

a. Testcase 1

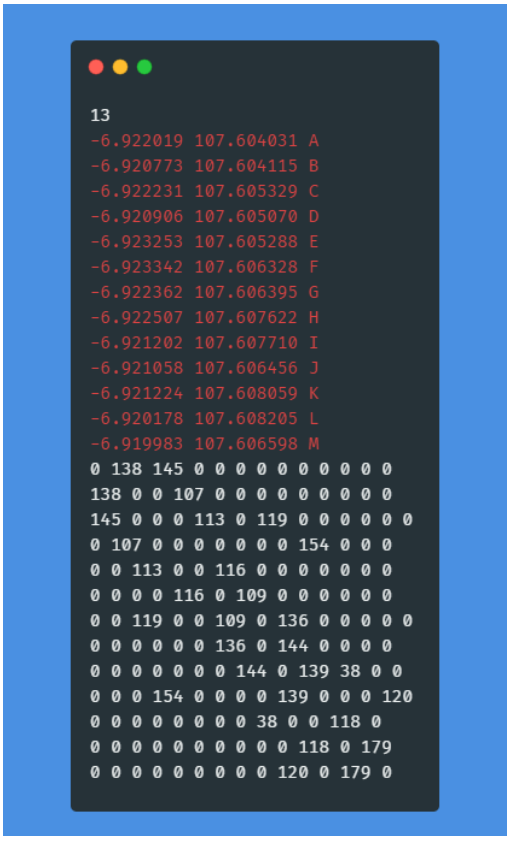
```
8
-6.892616 107.610423 Gerbang_Depan_ITB
-6.890464174304161 107.60724717865062 Kebun_Binatang_Bandung
-6.887407 107.611507 Gerbang_Belakang_ITB
-6.886107754151466 107.60814840083982 Sabuga_ITB
-6.886011891139484 107.61040145641657 Saraga_ITB
-6.884371565786224 107.61305147865777 McD_Dago
-6.893819336137485 107.61446768497895 RS_Borromeus
-6.890591990423812 107.60992938762139 Labtek5_ITB
0 428.35 0 0 0 0 419.13 233.94
428.35 0 584.79 446.03 0 0 0 0
0 584.79 0 427.57 193.57 0 0 398.50
0 446.03 427.57 0 310.81 0 0 0
0 0 193.57 310.81 0 339.15 0 0
0 0 0 0 339.15 0 1000.00 0
419.13 0 0 0 0 1000.00 0 0
233.94 0 398.50 0 0 0 0 0
```

Gambar 2.1 Testcase 1 (Sekitar ITB dan Dago)

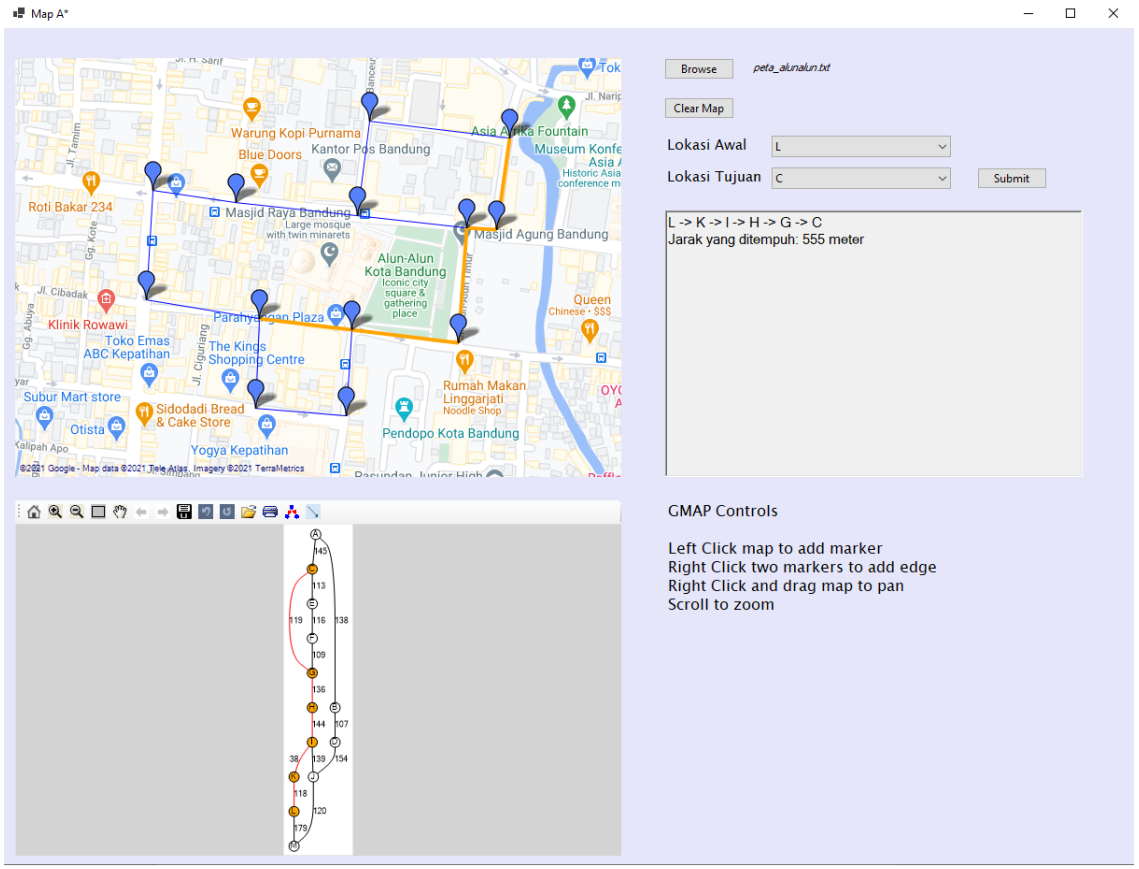


Gambar 2.2 Hasil Pencarian Testcase 1

b. Testcase 2

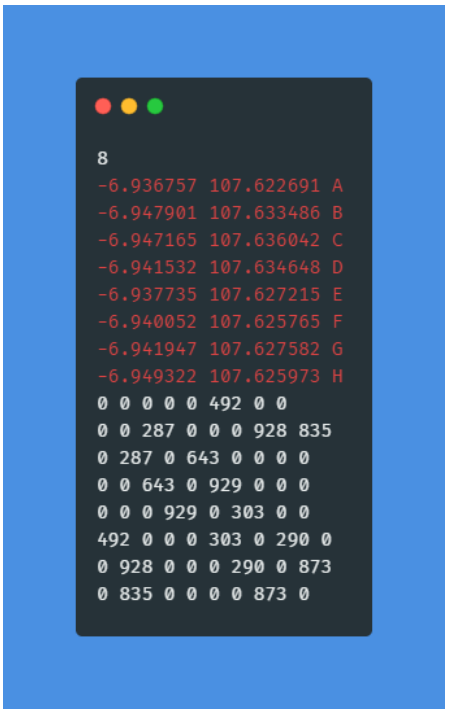


Gambar 2.3 Testcase 2 (Sekitar Alun-Alun Bandung)

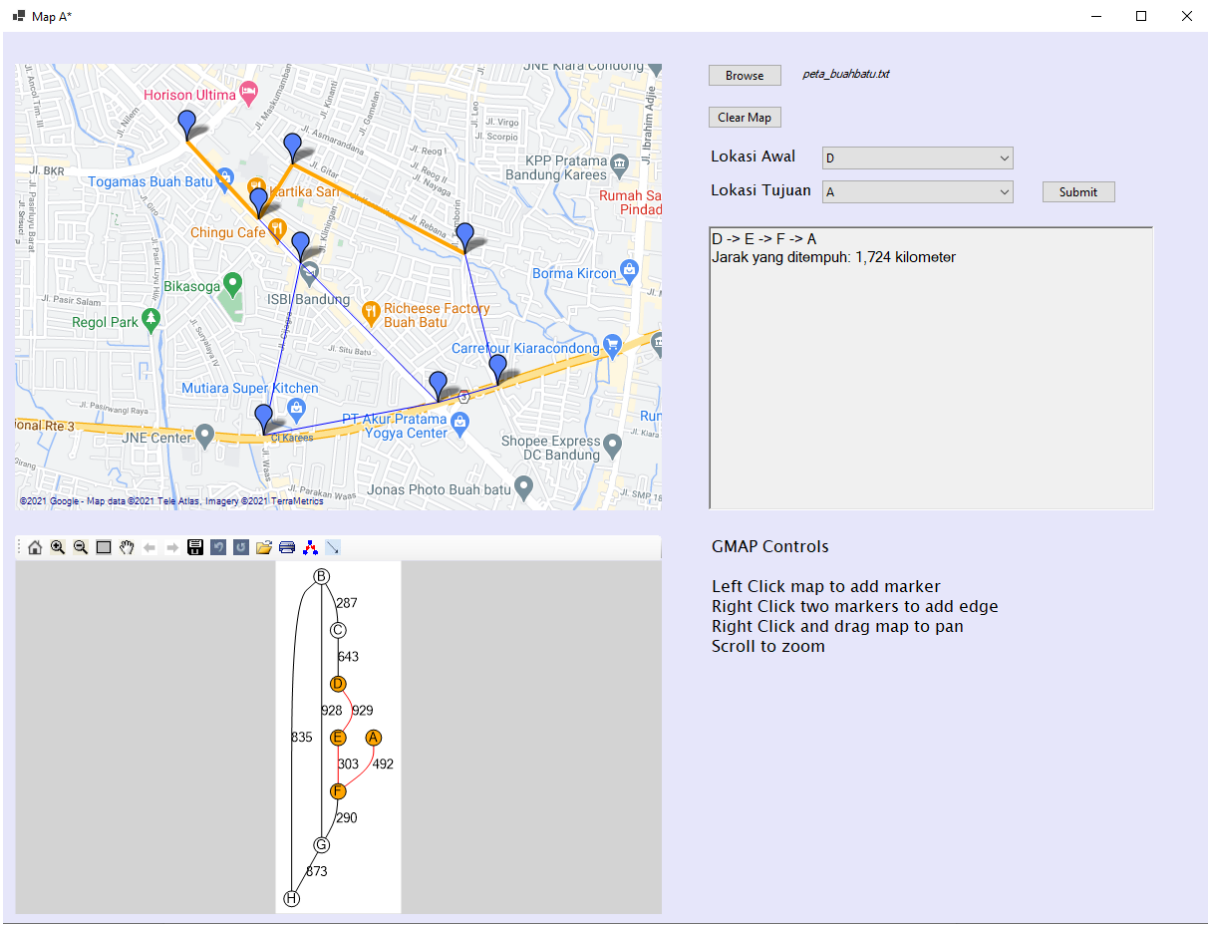


Gambar 2.4 Hasil Pencarian Testcase 2

c. Testcase 3



Gambar 2.5 Testcase 3 (Sekitar Buahbatu)

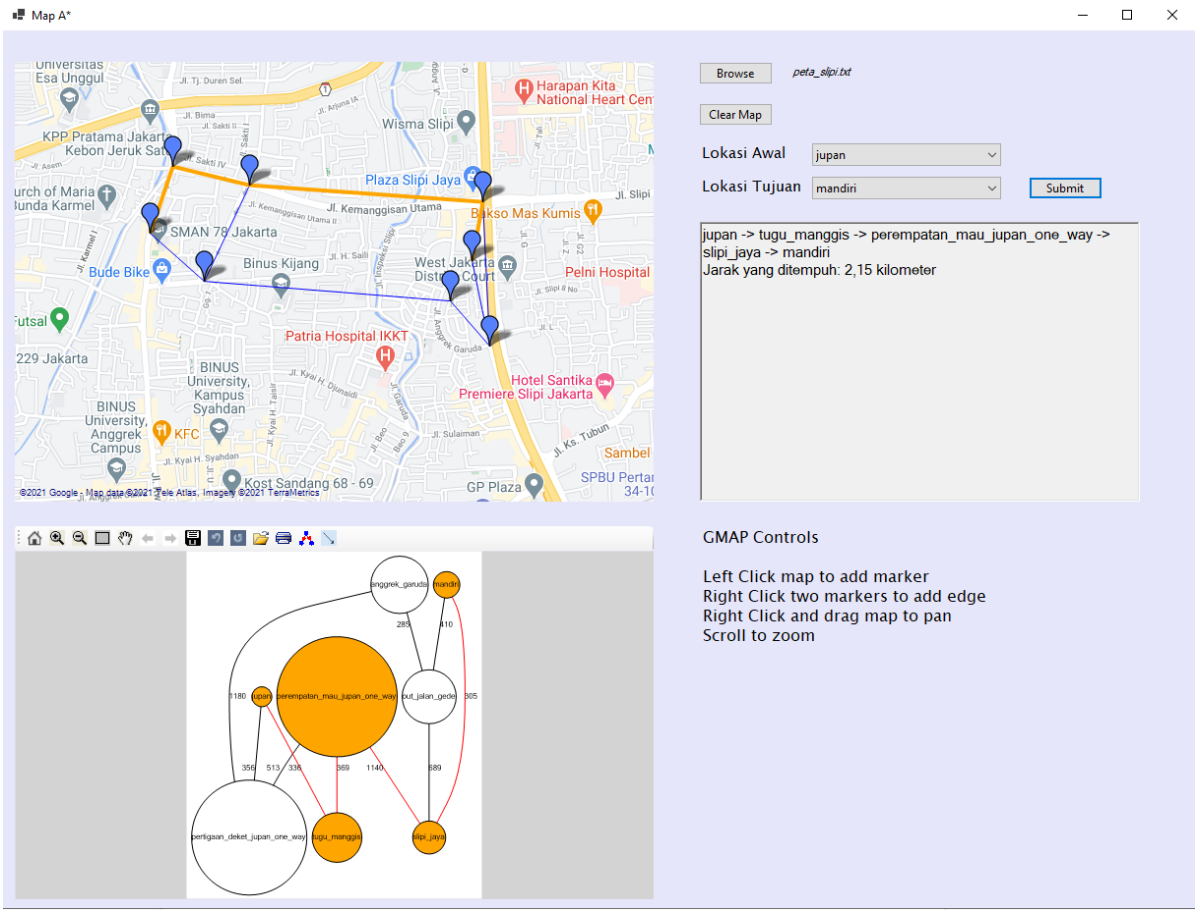


Gambar 2.6 Hasil Pencarian Testcase 3

d. Testcase 4

```
8
-6.189526591224791 106.79688933819102 slipi_jaya
-6.195788136889581 106.79721288032599 out_jalan_gede
-6.188818946104568 106.78673009811880 perempatan_mau_jupan_one_way
-6.192979027983100 106.78476727247056 pertigaan_deket_jupan_one_way
-6.190920434873802 106.78239462634383 jupan
-6.188004080867795 106.78338682385824 tugu_manggis
-6.192099838162099 106.79641480793924 mandiri
-6.193858216348896 106.79546574948853 anggrek_garuda
0 689 1140 0 0 0 305 0
689 0 0 0 0 0 410 285
1140 0 0 513 0 369 0 0
0 0 513 0 356 0 0 1180
0 0 0 356 0 336 0 0
0 0 369 0 336 0 0 0
305 410 0 0 0 0 0 0
0 285 0 1180 0 0 0 0
```

Gambar 2.7 Testcase 4 (Sekitar Wilayah Slipi, Jakarta Barat)

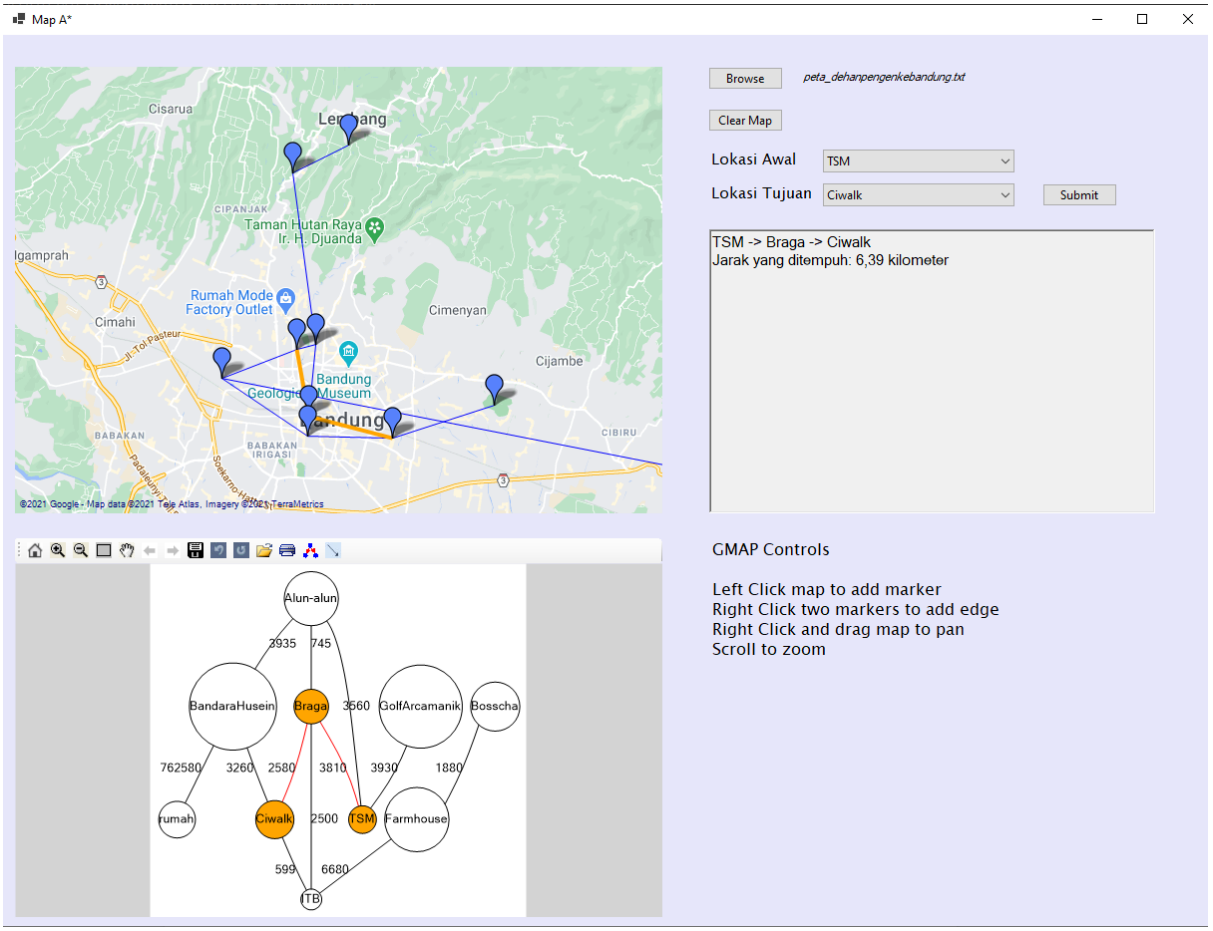


Gambar 2.8 Hasil Pencarian Testcase 4

e. Testcase 5

```
10
-8.221774990309962 114.3649748001016 rumah
-6.891714030635065 107.61009809705149 ITB
-6.923674515253688 107.63639590313785 TSM
-6.916182141558074 107.60750369468627 Braga
-6.922913576594546 107.6071499125641 Alun-alun
-6.903128737266028 107.57790392219113 BandaraHusein
-6.89347015731789 107.60355312640286 Ciwalk
-6.832938584357787 107.60219696251697 Farmhouse
-6.82333712602935 107.62141912854106 Bosscha
-6.912366371130943 107.67141225779218 GolfArcamanik
0 0 0 0 0 762580 0 0 0 0
0 0 0 2500 0 0 599 6680 0 0
0 0 0 3810 3560 0 0 0 0 3930
0 2500 3810 0 745 0 2580 0 0 0
0 0 3560 745 0 3935 0 0 0 0
762580 0 0 0 3935 0 3260 0 0 0
0 599 0 2580 0 3260 0 0 0 0
0 6680 0 0 0 0 0 0 1880 0
0 0 0 0 0 0 0 1880 0 0
0 0 3930 0 0 0 0 0 0 0
```

Gambar 2.9 Testcase 5 (Sekitar Bandung)

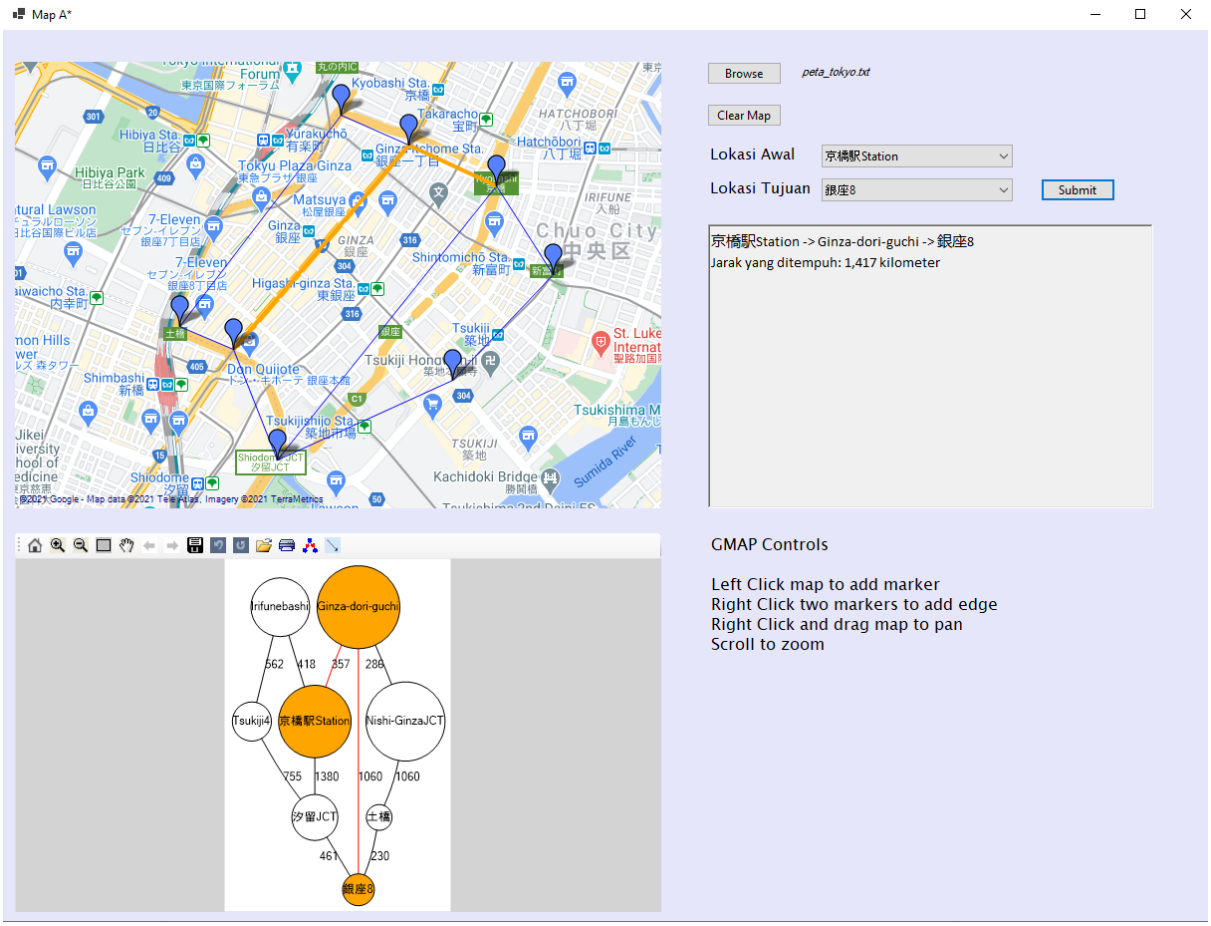


Gambar 2.10 Hasil Pencarian Testcase 5

f. Testcase 6

```
8
35.67580323260128 139.76582928709686 Nishi-GinzaJCT
35.66841975705375 139.75887429833233 土橋
35.667613092168935 139.7612108871603 銀座8
35.67477283123309 139.76877184211241 Ginza-dori-guchi
35.6733389193727 139.7725391336188 京橋駅Station
35.66374345633763 139.76307726811126 汐留JCT
35.666533056721406 139.77064660011897 Tsukiji4
35.670244503552865 139.7749876948699 Irifunebashi
0 1060 0 286 0 0 0 0
1060 0 230 0 0 0 0 0
0 230 0 1060 0 461 0 0
286 0 1060 0 357 0 0 0
0 0 0 357 0 1380 0 418
0 0 461 0 1380 0 755 0
0 0 0 0 0 755 0 562
0 0 0 0 418 0 562 0
```

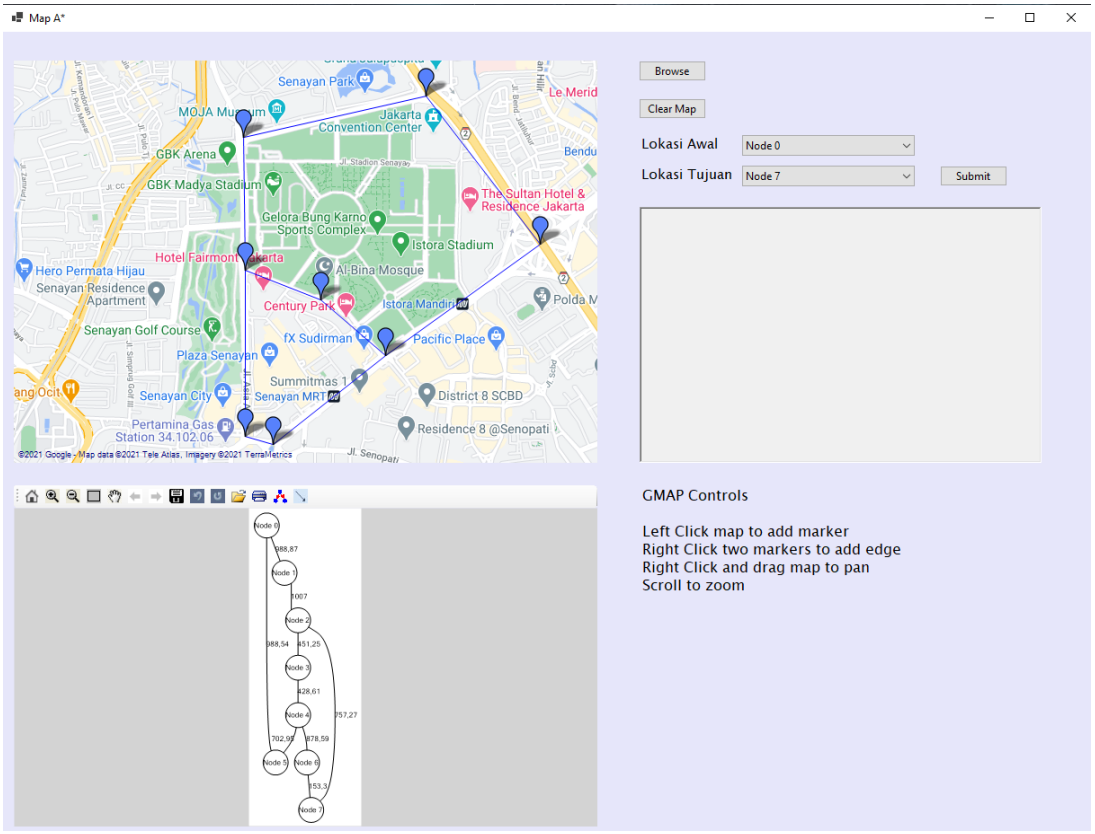
Gambar 2.11 Testcase 6 (Sekitar Ginza, Tokyo)



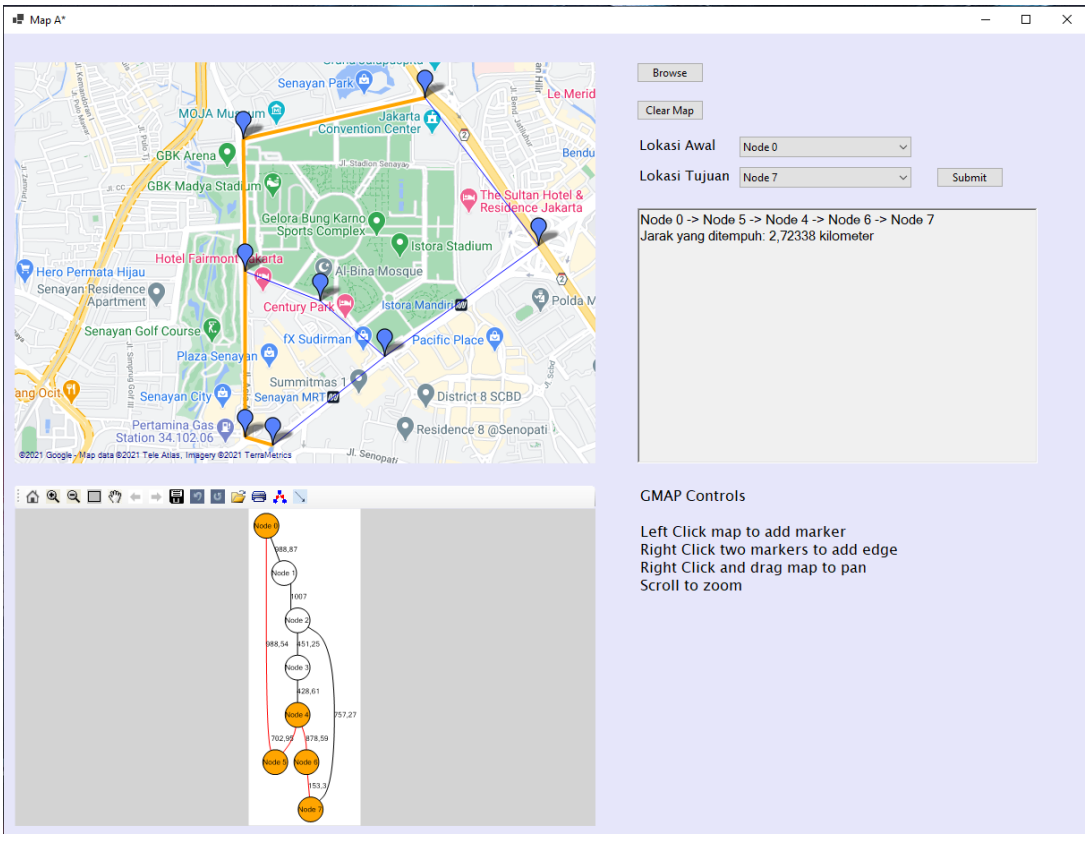
Gambar 2.12 Hasil Pencarian Testcase 6

g. Testcase Bonus

Pada testcase kali ini akan dites program dapat menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta (berupa jalan yang diberi warna). Simpul graf diperoleh dari peta (menggunakan API Google Map) dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Haversine.



Gambar 2.13 Testcase Bonus (Sekitar Gelora Bung Karno)



Gambar 2.14 Hasil Pencarian Testcase Bonus

3. Alamat Source Code

Source code dari program dapat diakses pada *link* berikut:

https://github.com/reymyr/Tucil3Stima

4. Tabel Cek List

No	Poin	Centang
1	Program dapat menerima input graf	√
2	Program dapat menghitung lintasan terpendek	√
3	Program dapat menampilkan lintasan terpendek serta jaraknya	√
4	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta	√