

TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA
SEMESTER II TAHUN 2022/2023
IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN
TERPENDEK



Disusun Oleh:

13521051 Manuella Ivana Uli Sianipar

13521059 Arleen Chrysantha Gunardi

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

DAFTAR ISI	2
DAFTAR GAMBAR	3
I. Deskripsi Persoalan	5
II. Kode Program	5
2.1 Coordinate	5
2.2 Vertex	6
2.3 Edge	7
2.4 Graph	8
2.5 FileReader	12
2.6 PriorityQueueCost	14
2.7 Solver	16
2.8 UCS	17
2.9 A-star	20
III. Pengujian Program	24
3.1 Pengujian 1: Peta Sekitar ITB	24
3.2 Pengujian 2: Peta Sekitar Alun-Alun Bandung	28
3.3 Pengujian 3: Peta Sekitar Buah Batu dan Bandung Selatan	32
3.4 Pengujian 4: Peta Daerah di Medan	36
3.5 Pengujian 5: Peta Sekitar Alun-Alun Bandung (Kasus Spesial)	41
3.6 Pengujian 6: Peta Sekitar ITB (Kasus Spesial)	45
3.7 Pengujian 7: File Tidak Valid	51
IV. Kesimpulan dan Komentar	52
V. Lampiran	52
4.1 Tautan Repositori GitHub	52
4.2 Tabel Penilaian	53
VI. Referensi	53

DAFTAR GAMBAR

Gambar 3.1.1 Peta Sekitar ITB	24
Gambar 3.1.2 Konfigurasi Graf Peta Sekitar ITB (map1.txt)	25
Gambar 3.1.3 Pengujian 1 dari ITB ke McD Dago (UCS)	26
Gambar 3.1.4 Pengujian 1 dari ITB ke McD Dago (A*)	26
Gambar 3.1.5 Pengujian 1 dari Kebun Binatang ke Unpad Dipatiukur (UCS)	27
Gambar 3.1.6 Pengujian 1 dari Kebun Binatang ke Unpad Dipatiukur (A*)	27
Gambar 3.1.7 Pengujian 1 dari ITHB ke ITB (UCS)	28
Gambar 3.1.8 Pengujian 1 dari ITHB ke ITB (A*)	28
Gambar 3.2.1 Peta Sekitar Alun-Alun Bandung	29
Gambar 3.2.2 Konfigurasi Graf Peta Sekitar Alun-Alun Bandung (map2.txt)	29
Gambar 3.2.3 Pengujian 2 dari Alun-Alun ke Kantor Pos (UCS)	30
Gambar 3.2.4 Pengujian 2 dari Alun-Alun ke Kantor Pos (A*)	30
Gambar 3.2.5 Pengujian 2 dari Taman Braga ke Menara BRI (UCS)	31
Gambar 3.2.6 Pengujian 2 dari Taman Braga ke Menara BRI (A*)	31
Gambar 3.3.1 Peta Sekitar Buah Batu dan Bandung Selatan	32
Gambar 3.3.2 Konfigurasi Graf Peta Sekitar Buah Batu dan Bandung Selatan (map3.txt)	32
Gambar 3.3.3 Pengujian 3 dari Transmart Buah Batu ke Rumah Sakit Mayapada (UCS)	33
Gambar 3.3.4 Pengujian 3 dari Transmart Buah Batu ke Rumah Sakit Mayapada (A*)	33
Gambar 3.3.5 Pengujian 3 dari Borma Kiaracondong ke Masjid Agung Buah Batu (UCS)	34
Gambar 3.3.6 Pengujian 3 dari Borma Kiaracondong ke Masjid Agung Buah Batu (A*)	35
Gambar 3.3.7 Pengujian 3 dari Telkom University ke ISBI (UCS)	35
Gambar 3.3.8 Pengujian 3 dari Telkom University ke ISBI (A*)	36
Gambar 3.4.1 Peta Daerah di Medan	36
Gambar 3.4.2 Konfigurasi Graf Peta Daerah di Medan (map4.txt)	37
Gambar 3.4.3 Pengujian 4 dari Pasar Petisah ke Columbia Asia Hospital (UCS)	38
Gambar 3.4.4 Pengujian 4 dari Pasar Petisah ke Columbia Asia Hospital (A*)	38
Gambar 3.4.5 Pengujian 4 dari Gramedia ke Brastagi Supermarket Tiara (UCS)	39
Gambar 3.4.6 Pengujian 4 dari Gramedia ke Brastagi Supermarket Tiara (A*)	39

Gambar 3.4.7 Pengujian 4 dari Sun Plaza ke Taman Ahmad Yani (UCS)	40
Gambar 3.4.8 Pengujian 4 dari Sun Plaza ke Taman Ahmad Yani (A*)	40
Gambar 3.5.1 Konfigurasi Graf Peta Sekitar Alun-Alun Bandung-Kasus Spesial(map2.1.txt)	41
Gambar 3.5.2 Pengujian 5 dari Alun-Alun ke Somewhere (UCS)	42
Gambar 3.5.3 Pengujian 5 dari Alun-Alun ke Somewhere (A*)	42
Gambar 3.5.4 Pengujian 5 dari Somewhere ke Yogya Kepatihan (UCS)	43
Gambar 3.5.5 Pengujian 5 dari Somewhere ke Yogya Kepatihan (A*)	43
Gambar 3.5.6 Pengujian 5 dari Alun-Alun ke Alun-Alun (UCS)	44
Gambar 3.5.7 Pengujian 5 dari Alun-Alun ke Alun-Alun(A*)	44
Gambar 3.6.1 Konfigurasi Graf Peta Sekitar ITB (map1.txt)	45
Gambar 3.6.2 Pengujian 6 dari ITB ke Kebun Binatang (UCS)	46
Gambar 3.6.3 Pengujian 6 dari Alun-Alun ke Somewhere (A*)	46
Gambar 3.6.4 Pengujian 6 dari Kebun Binatang ke ITB (UCS)	47
Gambar 3.6.5 Pengujian 6 dari Kebun Binatang ke ITB (A*)	47
Gambar 3.6.6 Konfigurasi Graf Peta Sekitar ITB Kasus Khusus (map1.1.txt)	48
Gambar 3.6.7 Pengujian 6 dari ITB ke Kebun Binatang (UCS)	49
Gambar 3.6.8 Pengujian 6 dari Alun-Alun ke Somewhere (A*)	49
Gambar 3.6.9 Pengujian 6 dari Kebun Binatang ke ITB (UCS)	50
Gambar 3.6.10 Pengujian 6 dari Kebun Binatang ke ITB (A*)	50
Gambar 3.7.1 Konfigurasi Graf Peta Tidak Valid (map5.txt)	51
Gambar 3.7.2 Pengujian 7 File Tidak Valid	52

I. Deskripsi Persoalan

Implementasi algoritma UCS (Uniform Cost Search) dan A* dapat digunakan untuk menentukan lintasan terpendek antara dua titik lokasi. Pada Tugas Kecil 3 ini, dibuatlah suatu program untuk menentukan lintasan terpendek antara dua lokasi pada peta di kehidupan nyata. Peta tersebut berbentuk graf yang memuat titik-titik lokasi, jalan-jalan yang dapat dilalui, serta persimpangan jalan. Peta berupa graf berbobot dengan bobot merepresentasikan jarak antara dua titik lokasi. Jarak tersebut dihitung dengan menggunakan ruler pada Google Maps.

Program ini dibuat dengan memanfaatkan bahasa pemrograman C# dan *framework* .NET. Program akan meminta masukan *file* .txt peta berupa graf berbobot yang terdiri dari jumlah simpul, detail simpul (nama lokasi, koordinat *latitude* dan *longitude*), serta matriks ketetanggaan. Matriks ketetanggaan berupa matriks integer yang menyatakan jarak antara dua simpul dalam satuan meter. Setelah memasukkan *file* peta, pengguna juga diminta untuk memilih simpul lokasi asal dan simpul lokasi tujuan, kemudian algoritma pencarinya (UCS atau A*). Keluaran dari program adalah jarak terpendek dari lokasi asal ke lokasi tujuan beserta dengan visualisasi graf dan simpul-simpul yang dibangkitkan selama pencarian.

II. Kode Program

Implementasi program dirancang dengan konsep pemrograman berorientasi objek. Terdapat beberapa kelas yang didefinisikan untuk merancang program, antara lain:

2.1 Coordinate

Kelas Coordinate menyatakan koordinat suatu lokasi pada peta yang direpresentasikan dengan *latitude* dan *longitude*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    public class Coordinate
    {
        /* attributes */
    }
}
```

```

    public double latitude { get; set; } // row
    public double longitude { get; set; } // col

    /* ctor */
    public Coordinate(double _latitude, double _longitude)
    {
        latitude = _latitude;
        longitude = _longitude;
    }
}

```

2.2 Vertex

Kelas Vertex menyatakan simpul yang terdapat pada graf. Simpul didefinisikan dengan nama lokasinya serta koordinat lokasinya. Untuk memudahkan proses implementasi algoritma UCS, dibuat kelas VertexPathCost yang menyimpan simpul saat ini, *path* berupa *list of Vertex*, dan *cost* berupa jarak dari titik asal sampai ke simpul yang sedang dievaluasi saat ini.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    public class Vertex
    {
        /* attributes */
        public String locName { get; set; }
        public Coordinate coordinate { get; set; }

        /* ctor */
        public Vertex(String locName, Coordinate coordinate)
        {
            this.locName = locName;
        }
    }
}

```

```

        this.coordinate = coordinate;
    }
}

public class VertexPathCost
{
    /* attributes */
    public Vertex vertex { get; set; }
    public List<VertexPathCost> path { get; set; }
    public double cost { get; set; }
    /* ctor */
    public VertexPathCost(Vertex vertex, List<VertexPathCost> path, double
cost)
    {
        this.vertex = vertex;
        this.path = path;
        this.cost = cost;
    }
    public VertexPathCost(Vertex vertex)
    {
        this.vertex = vertex;
        this.path = new List<VertexPathCost>() { };
        this.cost = 0;
    }
}
}

```

2.3 Edge

Kelas Edge menyatakan sisi antara dua simpul dengan bobot tertentu.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    public class Edge
    {

```

```

/* attributes */
public Vertex V1 { get; set; }
public Vertex V2 { get; set; }
public double weight { get; set; } // distance between V1 and V2 in meters

/* ctor */
public Edge(Vertex V1, Vertex V2, double weight)
{
    this.V1 = V1;
    this.V2 = V2;
    this.weight = weight;
}
}

```

2.4 Graph

Kelas Graph menyatakan graf yang terdiri atas simpul-simpul, sisi antara simpul-simpul tersebut, dan matriks ketetanggaannya.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    public class Graph
    {
        /* attributes */
        public int vertexCount { get; set; }
        public List<Vertex> vertices { get; set; }
        public List<Edge> edges { get; set; }
        public double[,] adjacencyMatrix { get; set; }

        /* ctor */
        public Graph()
        {
            vertexCount = 0;
            vertices = new List<Vertex>();
        }
    }
}

```

```

        edges = new List<Edge>();
        adjacencyMatrix = null;
    }
    public Graph(int vertexCount, List<Vertex> vertices, List<Edge> edges,
double[,] adjacencyMatrix)
{
    this.vertexCount = vertexCount;
    this.vertices = vertices;
    this.edges = edges;
    this.adjacencyMatrix = adjacencyMatrix;
}

/* methods */
public int findIndex(Vertex v)
{
    bool found = false;
    int i = 0;
    while (i < vertexCount && !found)
    {
        if (vertices[i] == v)
        {
            found = true;
        }
        else
        {
            i++;
        }
    }
    if (found)
    {
        return i;
    }
    else
    {
        return -1;
    }
}

/* method */
public Boolean adaJalan(Vertex v1, Vertex v2)
{

```

```

// cari index v1 dan v2
int i = findIndex(v1);
int j = findIndex(v2);

if (adjacencyMatrix != null && adjacencyMatrix[i, j] > 0)
{
    return true;
}
else
{
    return false;
}

public double getWeight(Vertex v1, Vertex v2)
{
    bool found = false;
    int i = 0;
    while (i < edges.Count && !found)
    {
        if (edges[i].V1 == v1 && edges[i].V2 == v2)
        {
            found = true;
        }
        else
        {
            i++;
        }
    }
    if (found)
    {
        return edges[i].weight;
    }
    else
    {
        return -1;
    }
}

public List<Vertex> getNeighbour(Vertex v)
{

```

```

        int i = findIndex(v);
        List<Vertex> arr = new List<Vertex>();
        for (int j = 0; j < vertexCount; j++)
        {
            if (adjacencyMatrix[i, j] > 0)
            {
                arr.Add(vertices[j]);
            }
        }
        return arr;
    }

    public Vertex getVertex(String name)
    {
        bool found = false;
        int i = 0;
        while(i < vertices.Count && !found)
        {
            if (vertices[i].locName == name)
            {
                found = true;
            }
            else
            {
                i++;
            }
        }
        if (found)
        {
            return vertices[i];
        }
        else
        {
            return new Vertex("notfound", new Coordinate(0,0));
        }
    }
}

```

2.5 FileReader

Kelas FileReader merupakan kelas berisi metode untuk membaca peta dari *file* dan menerjemahkannya menjadi graf peta.

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.TreeView;

namespace PathFinder
{
    public class FileReader
    {
        public Graph readFile(String filename)
        {
            Graph graph = new Graph();

            string pathfilename = "..\\..\\..\\..\\bin\\" + filename;

            using (StreamReader reader = new StreamReader(pathfilename))
            {
                string line = reader.ReadLine();
                if (line == null) throw new InvalidDataException("File tidak valid!");

                // vertex count
                int vertexCount;
                var Result = int.TryParse(line, out vertexCount);
                if (!Result) throw new InvalidDataException("File tidak valid!");
                graph.vertexCount = vertexCount;

                // vertices
                int vCount = 0;
                line = reader.ReadLine();
                while (vCount < graph.vertexCount && line != null)
                {
                    string[] vertex = line.Split(' ');
                    graph.vertices.Add(vertex[0]);
                    graph.degrees.Add(0);
                    graph.neighbors.Add(vertex[0], new Dictionary<String, String>());
                    vCount++;
                }
            }
        }
    }
}
```

```

        if (vertex.Length != 3) throw new InvalidDataException("File
tidak valid!");
            string locName = vertex[0];
            double coordinateX = Convert.ToDouble(vertex[1],
CultureInfo.InvariantCulture);
            double coordinateY = Convert.ToDouble(vertex[2],
CultureInfo.InvariantCulture);
            graph.vertices.Add(new Vertex(locName, new
Coordinate(coordinateX, coordinateY)));

            line = reader.ReadLine();
            vCount++;
        }

        // adjacency matrix
        vCount = 0;
        graph.adjacencyMatrix = new double[graph.vertexCount,
graph.vertexCount];
        while (vCount < graph.vertexCount && line != null)
        {
            string[] matrixRow = line.Split(' ');
            if (matrixRow.Length != graph.vertexCount) throw new
InvalidDataException("File tidak valid");

            for (int col = 0; col < graph.vertexCount; col++)
            {
                graph.adjacencyMatrix[vCount, col] =
int.Parse(matrixRow[col]);
            }

            line = reader.ReadLine();
            vCount++;
        }

        // edges
        // list of edges between vertex 1 and vertex 2 (with value 1 in
adjacency matrix)
        // weight calculated by haversine formula between two vertex
coordinates (longitude and latitude)
        // precondition: adjacency matrix is symmetrical
        for (int i = 0; i < graph.adjacencyMatrix.GetLength(0); i++)

```

```

    {
        for (int j = 0; j < graph.adjacencyMatrix.GetLength(1); j++)
        {
            if (graph.adjacencyMatrix[i, j] != 0)
            {
                Vertex V1 = graph.vertices[i];
                Vertex V2 = graph.vertices[j];
                graph.edges.Add(new Edge(V1, V2,
graph.adjacencyMatrix[i, j]));
            }
        }
        return graph;
    }
}
}

```

2.6 PriorityQueueCost

Kelas PriorityQueueCost merupakan kelas *priority queue* berdasarkan *cost* suatu simpul. Jenis elemen pada PriorityQueueCost adalah VertexPathCost.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    class PriorityQueueCost
    {
        // Priority Queue of VertexPathCost, ordered by cost (ascending)
        public List<VertexPathCost> queue { get; set; }

        /* ctor */
        public PriorityQueueCost()
        {
            queue = new List<VertexPathCost>() { };
        }
    }
}

```

```

public PriorityQueueCost(List<VertexPathCost> queue)
{
    this.queue = queue;
}

/* methods */
public VertexPathCost Peek()
{
    // get the head of the queue (0th element)
    return queue[0];
}

public int Count()
{
    // get the number of elements in queue
    return queue.Count;
}

public void Enqueue(VertexPathCost v)
{
    // enqueue with priority (cost priority, ascending)
    if (queue.Count == 0)
    {
        queue.Add(v); // if queue is empty
    }
    else
    {
        // insert into queue according to its cost
        bool isFound = false;
        int idx = queue.Count - 1;
        while (idx > 0 && !isFound)
        {
            if (v.cost < queue[idx].cost)
            {
                idx--;
            }
            else
            {
                isFound = true;
            }
        }
    }
}

```

```

        if (idx == 0)
        {
            if (v.cost < queue[idx].cost)
            {
                queue.Insert(idx, v);
            }
            else
            {
                queue.Insert(idx + 1, v);
            }
        }
        else if (idx == queue.Count - 1)
        {
            queue.Add(v);
        }
        else
        {
            queue.Insert(idx + 1, v);
        }
    }
}

public void Dequeue()
{
    // remove 0th element of the queue
    queue.RemoveAt(0);
}
}
}

```

2.7 Solver

Kelas Solver merupakan *superclass* dari kelas UCS dan A-star. Atribut kelas ini adalah hal-hal yang dibutuhkan ketika melakukan pencarian rute terpendek, yaitu graf peta, simpul awal dan tujuan, jarak terpendek, dan juga rute solusi.

```

using PathFinder;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace PathFinder
{
    public class Solver
    {
        /* attributes */
        public Graph graph { get; set; }
        public Vertex start { get; set; }
        public Vertex end { get; set; }
        public double distance { get; set; }
        public List<VertexPathCost> solution { get; set; }
        public List<(VertexPathCost, List<VertexPathCost>)> history { get; set; }

        /* ctor */
        public Solver(Graph g, Vertex s, Vertex e)
        {
            graph = g;
            start = s;
            end = e;
            distance = 0;
            solution = new List<VertexPathCost>();
            history = new List<(VertexPathCost, List<VertexPathCost>)>() { };
        }
    }
}

```

2.8 UCS

Kelas UCS merupakan turunan dari kelas Solver. Kelas ini memiliki atribut tambahan *history* bertipe *list of VertexPathCost* yang merepresentasikan sekumpulan simpul yang dibangkitkan selama pencarian berlangsung. Selain itu, kelas UCS juga memiliki sebuah metode untuk mencari rute terpendek.

```

using PathFinder;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace PathFinder
{
    public class UCS : Solver
    {
        public UCS(Graph g, Vertex s, Vertex e) : base(g, s, e) { }

        public void UCSSolver()
        {
            /* initialize */
            solution = new List<VertexPathCost>() { };
            distance = 0;
            PriorityQueueCost prioqueue = new PriorityQueueCost();
            VertexPathCost currentVertex = new VertexPathCost(start, new
List<VertexPathCost>() { }, distance);
            Boolean isArrived = (currentVertex.vertex == end);
            Boolean[] isVisited = new Boolean[graph.vertexCount];
            double cost;
            history = new List<(VertexPathCost, List<VertexPathCost>)>() { };
            List<VertexPathCost> historyQueue = new List<VertexPathCost>() { };

            // Add vertices adjacent with start vertex to prioqueue
            for (int j = 0; j < graph.vertexCount; j++)
            {
                if (graph.adaJalan(start, graph.vertices[j]))
                {
                    cost = graph.getWeight(start, graph.vertices[j]);
                    prioqueue.Enqueue(new VertexPathCost(graph.vertices[j], new
List<VertexPathCost>() { currentVertex }, cost));
                    historyQueue.Add(new VertexPathCost(graph.vertices[j], new
List<VertexPathCost>() { currentVertex }, cost));
                }
            }

            history.Add((currentVertex, historyQueue)); // add to history
            isVisited[graph.findIndex(start)] = true; // marked as visited

            // Search while hasn't arrived at the destination and prioqueue is not
empty
            while (!isArrived && prioqueue.Count() > 0)
            {

```

```

    // Dequeue
    currentVertex = prioqueue.Peek();
    prioqueue.Dequeue();
    isVisited[graph.findIndex(currentVertex.vertex)] = true;

    // check if has reached the destination
    if (currentVertex.vertex == end)
    {
        isArrived = true;
        currentVertex.path.Add(currentVertex);
        history.Add((currentVertex, new List<VertexPathCost>() { }));
        break;
    }

    // enqueue
    historyQueue = new List<VertexPathCost>() { };
    for (int j = 0; j < graph.vertexCount; j++)
    {
        if (!isVisited[j] && graph.adaJalan(currentVertex.vertex,
graph.vertices[j]))
        {
            List<VertexPathCost> currentPath = new
List<VertexPathCost>(currentVertex.path);
            if (currentPath.Last() != currentVertex)
            {
                currentPath.Add(currentVertex);
            }
            cost = currentVertex.cost +
graph.getWeight(currentVertex.vertex, graph.vertices[j]);
            prioqueue.Enqueue(new VertexPathCost(graph.vertices[j],
currentPath, cost));
            historyQueue.Add(new VertexPathCost(graph.vertices[j],
currentPath, cost));
        }
    }
    history.Add((currentVertex, historyQueue)); // add to history
}

if (prioqueue.Count() == 0 && !isArrived)
{
    // path tidak ditemukan
}

```

```

        distance = 0;
        solution = new List<VertexPathCost>() { };
    }
    else
    {
        // result
        if (currentVertex.vertex == start)
        {
            currentVertex.path.Add(currentVertex);
        }
        solution = currentVertex.path;
        distance = currentVertex.cost;
    }
}
}
}

```

2.9 A-star

Kelas A-star merupakan kelas turunan dari kelas Solver.

```

using PathFinder;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PathFinder
{
    public class Astar : Solver
    {
        public Astar(Graph g, Vertex s, Vertex e) : base(g, s, e) { }

        public double straightDist(Vertex v1, Vertex v2)
        {
            double v1lat = v1.coordinate.latitude * (Math.PI / 180);
            double v2lat = v2.coordinate.latitude * (Math.PI / 180);
            double difflon = (v1.coordinate.longitude - v2.coordinate.longitude) *
(Math.PI / 180);
        }
    }
}

```

```

        double a = Math.Pow(Math.Sin((v1lat - v2lat) / 2), 2) +
Math.Cos(v1lat) * Math.Cos(v2lat) * Math.Pow(Math.Sin(difflon / 2), 2);
        double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
        double d = 6378137 * c;
        return d;
    }

    public void AstarSolver()
    {
        /* initialize */
        solution = new List<VertexPathCost>() { };
        distance = 0;
        PriorityQueueCost prioqueue = new PriorityQueueCost();
        VertexPathCost currentVertex = new VertexPathCost(start, new
List<VertexPathCost>() { }, distance);
        Boolean isArrived = (currentVertex.vertex == end);
        Boolean[] isVisited = new Boolean[graph.vertexCount];
        double cost;
        history = new List<(VertexPathCost, List<VertexPathCost>)>() { };
        List<VertexPathCost> historyQueue = new List<VertexPathCost>() { };

        // Add vertices adjacent with start vertex to prioqueue
        for (int j = 0; j < graph.vertexCount; j++)
        {
            if (graph.adajalan(start, graph.vertices[j]))
            {
                cost = graph.getWeight(start, graph.vertices[j]) +
straightDist(graph.vertices[j], end);
                prioqueue.Enqueue(new VertexPathCost(graph.vertices[j], new
List<VertexPathCost>() { currentVertex }, cost));
                historyQueue.Add(new VertexPathCost(graph.vertices[j], new
List<VertexPathCost>() { currentVertex }, cost));
            }
        }
        history.Add((currentVertex, historyQueue)); // add to history
        isVisited[graph.findIndex(start)] = true; // marked as visited

        // Search while hasn't arrived at the destination and prioqueue is not
empty
        while (!isArrived && prioqueue.Count() > 0)
    }
}

```

```

{
    // Dequeue
    currentVertex = prioqueue.Peek();
    prioqueue.Dequeue();
    isVisited[graph.findIndex(currentVertex.vertex)] = true;

    // check if has reached the destination
    if (currentVertex.vertex == end)
    {
        isArrived = true;
        currentVertex.path.Add(currentVertex);
        history.Add((currentVertex, new List<VertexPathCost>() { }));
        break;
    }

    // enqueue
    historyQueue = new List<VertexPathCost>() { };
    for (int j = 0; j < graph.vertexCount; j++)
    {
        if (!isVisited[j] && graph.adaJalan(currentVertex.vertex,
graph.vertices[j]))
        {
            List<VertexPathCost> currentPath = new
List<VertexPathCost>(currentVertex.path);
            if (currentPath.Last() != currentVertex)
            {
                currentPath.Add(currentVertex);
            }
            cost = currentVertex.cost +
graph.getWeight(currentVertex.vertex, graph.vertices[j]) +
straightDist(graph.vertices[j], end);
            prioqueue.Enqueue(new VertexPathCost(graph.vertices[j],
currentPath, cost));
            historyQueue.Add(new VertexPathCost(graph.vertices[j],
currentPath, cost));
        }
    }
    history.Add((currentVertex, historyQueue)); // add to history
}

if (prioqueue.Count() == 0 && !isArrived)

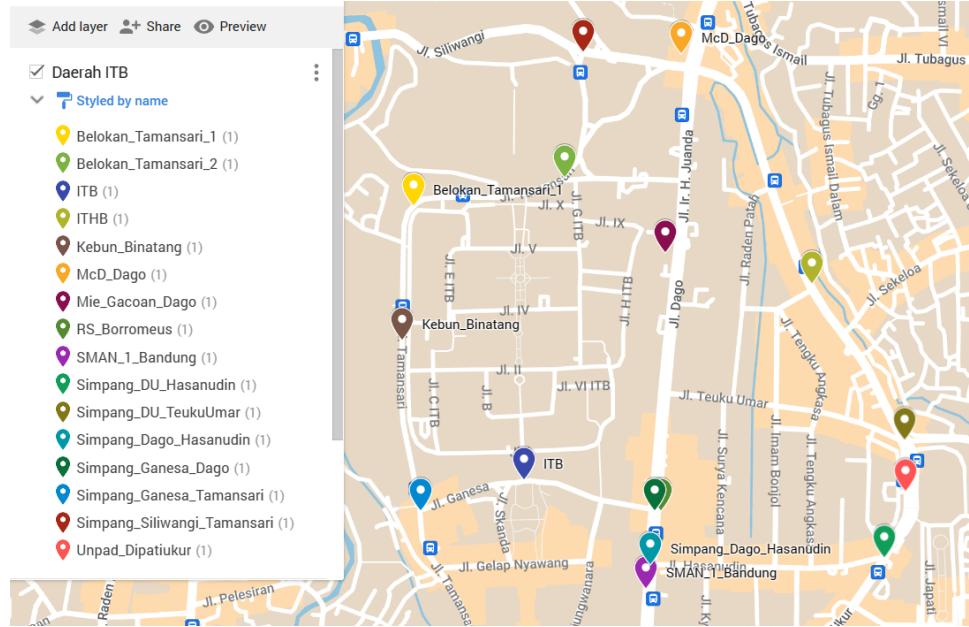
```

```
{  
    // path tidak ditemukan  
    distance = 0;  
    solution = new List<VertexPathCost>() { };  
}  
else  
{  
    // result  
    if (currentVertex.vertex == start)  
    {  
        currentVertex.path.Add(currentVertex);  
    }  
    solution = currentVertex.path;  
    distance = 0;  
    Vertex v1;  
    Vertex v2;  
    for (int i = 0; i < solution.Count - 1; i++)  
    {  
        v1 = solution[i].vertex;  
        v2 = solution[i + 1].vertex;  
        distance += graph.getWeight(v1, v2);  
    }  
}
```

III. Pengujian Program

3.1 Pengujian 1: Peta Sekitar ITB

Input peta: map1.txt



Gambar 3.1.1 Peta Sekitar ITB

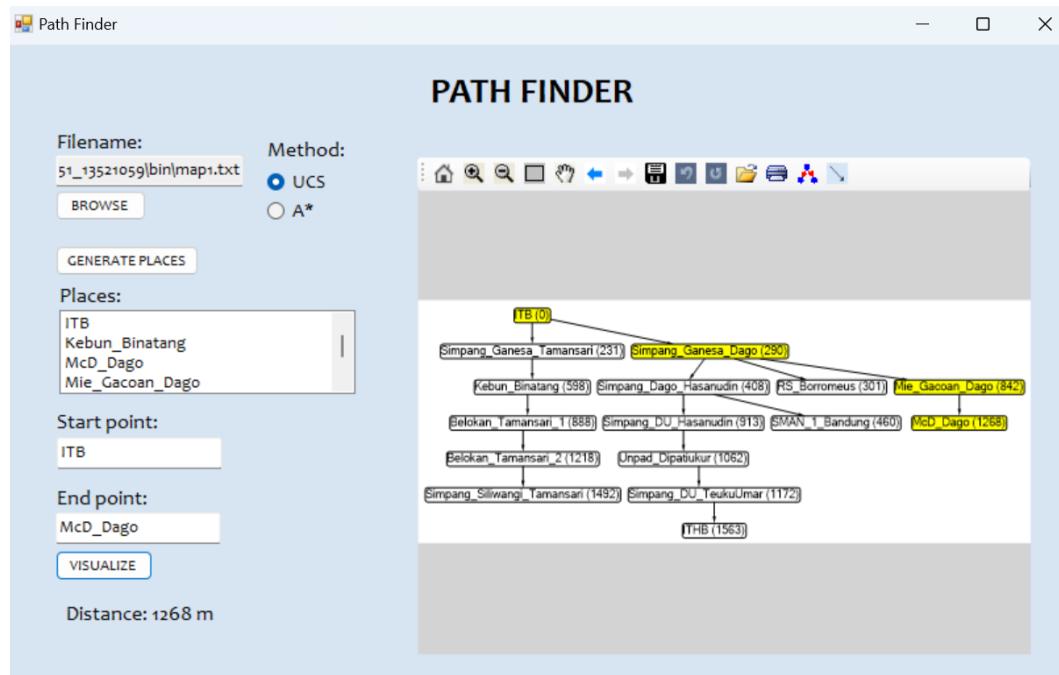
```
map1.txt
File Edit View
16
ITB -6.89315 107.61042
Kebun_Binatang -6.89046 107.60806
McD_Dago -6.88496 107.61349
Mie_Gacoan_Dago -6.88878 107.61316
RS_Borromeus -6.89374 107.61307
SMAN_1_Bandung -6.89525 107.6128
Unpad_Dipatiukur -6.89338 107.61783
ITHB -6.88939 107.61601
Simpang_Ganesa_Dago -6.89374 107.61297
Simpang_Ganesa_Tamansari -6.89374 107.60842
Simpang_Siliwangi_Tamansari -6.8849 107.61157
Belokan_Tamansari_1 -6.88787 107.60828
Belokan_Tamansari_2 -6.88733 107.61121
Simpang_Dago_Hasanudin -6.89479 107.61288
Simpang_DU_Hasanudin -6.89466 107.61744
Simpang_DU_TeukuUmar -6.89239 107.61782
000 000 000 000 000 000 000 000 290 231 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 367 000 290 000 000 000 000 000 000 000 000 000 000
000 000 000 426 000 000 000 564 000 000 215 000 000 000 000 000 000 000 000 000 000 000
000 000 426 000 000 000 000 000 552 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 011 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 052 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 149 110
000 000 564 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 391
290 000 000 552 011 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 118 000 000
231 367 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 215 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 274 000 000 000
000 290 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 330 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
Ln 16, Col 40 100% Windows (CRLF) UTF-8
```

Gambar 3.1.2 Konfigurasi Graf Peta Sekitar ITB (map1.txt)

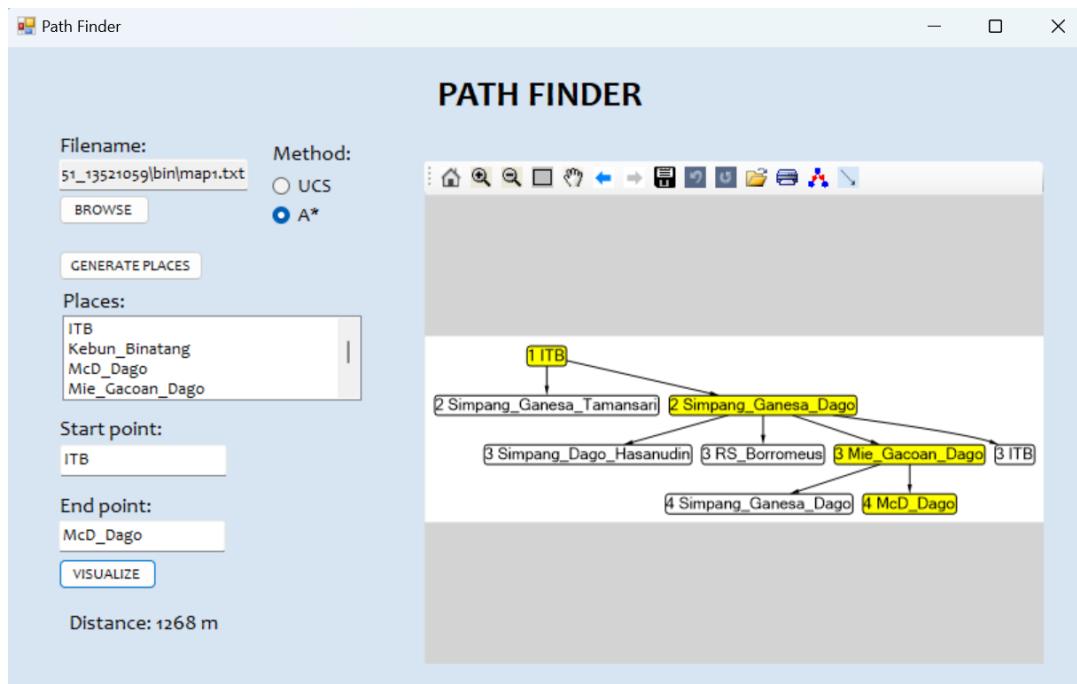
Pencarian:

Start : ITB

End : McD_Dago

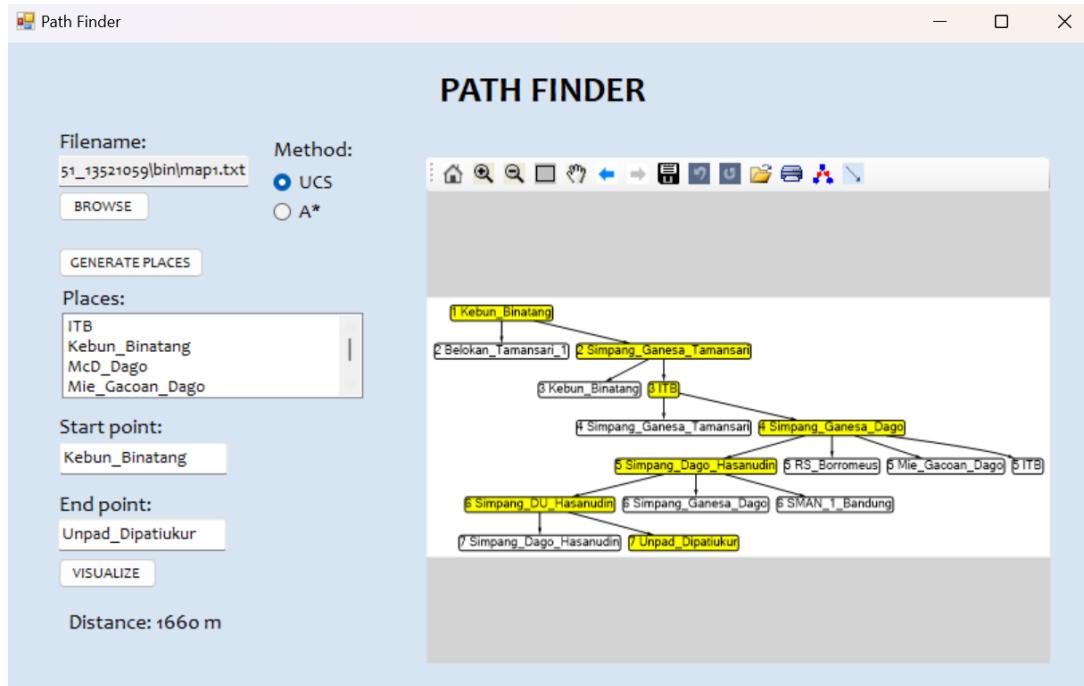


Gambar 3.1.3 Pengujian 1 dari ITB ke McD Dago (UCS)

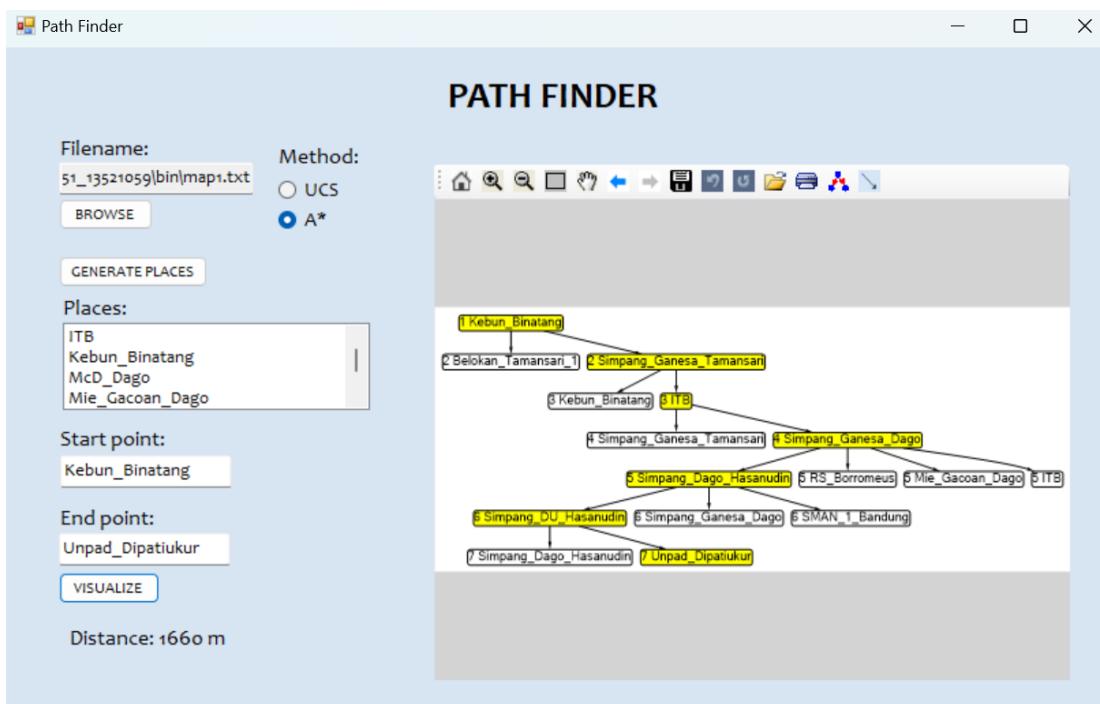


Gambar 3.1.4 Pengujian 1 dari ITB ke McD Dago (A*)

Start : Kebun_Binatang
End : Unpad_Dipatiukur



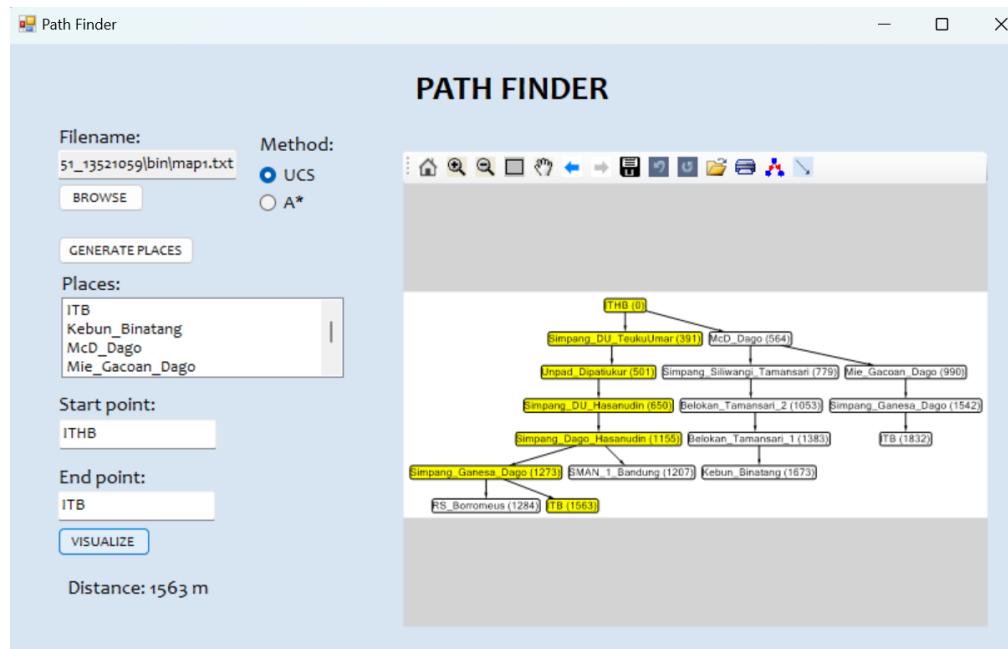
Gambar 3.1.5 Pengujian 1 dari Kebun Binatang ke Unpad Dipatiukur (UCS)



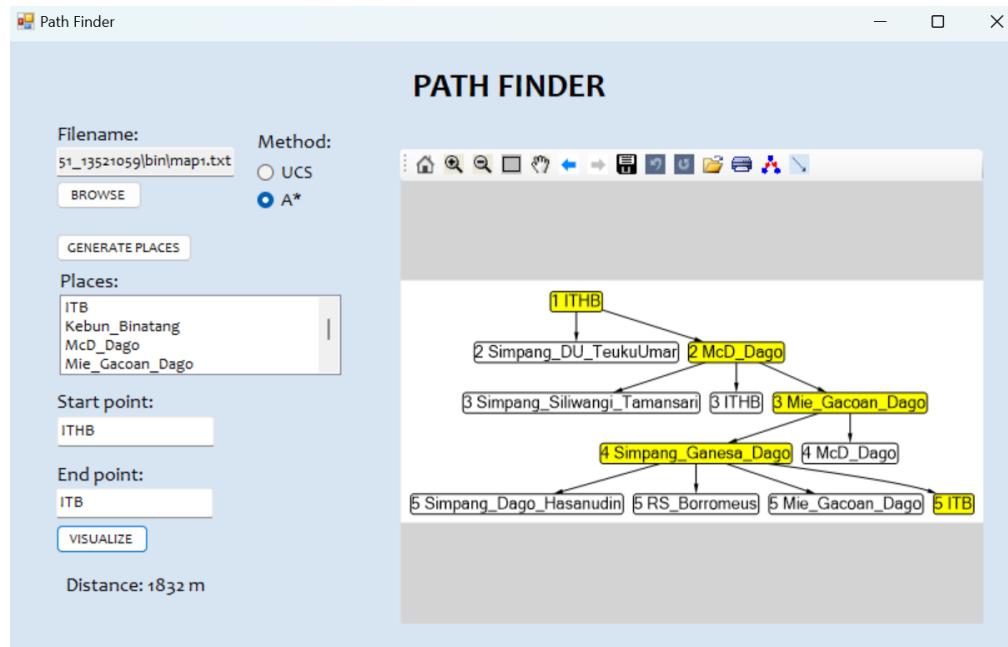
Gambar 3.1.6 Pengujian 1 dari Kebun Binatang ke Unpad Dipatiukur (A*)

Start : ITHB

End : ITB



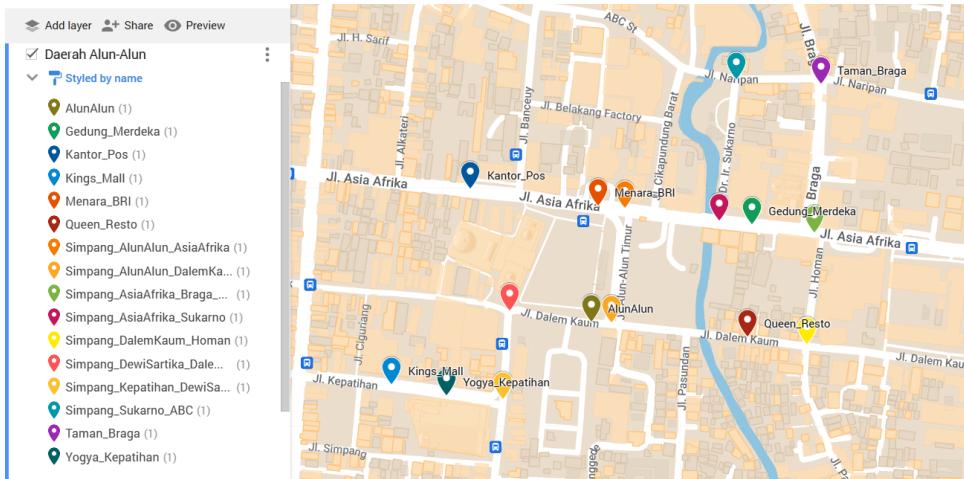
Gambar 3.1.7 Pengujian 1 dari ITHB ke ITB (UCS)



Gambar 3.1.8 Pengujian 1 dari ITHB ke ITB (A*)

3.2 Pengujian 2: Peta Sekitar Alun-Alun Bandung

Input peta: map2.txt



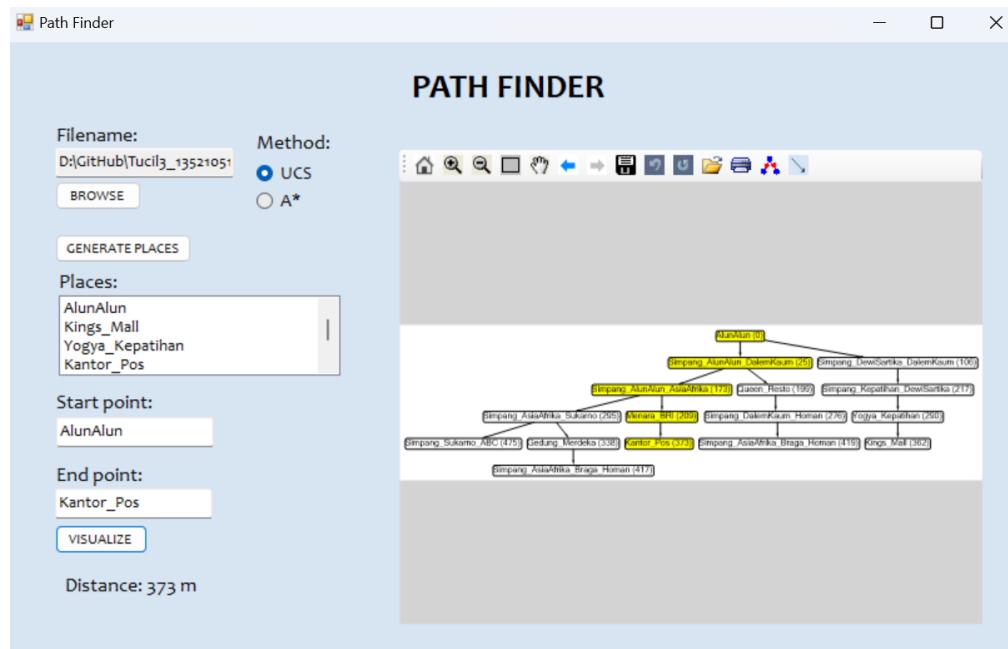
Gambar 3.2.1 Peta Sekitar Alun-Alun Bandung

Gambar 3.2.2 Konfigurasi Graf Peta Sekitar Alun-Alun Bandung (map2.txt)

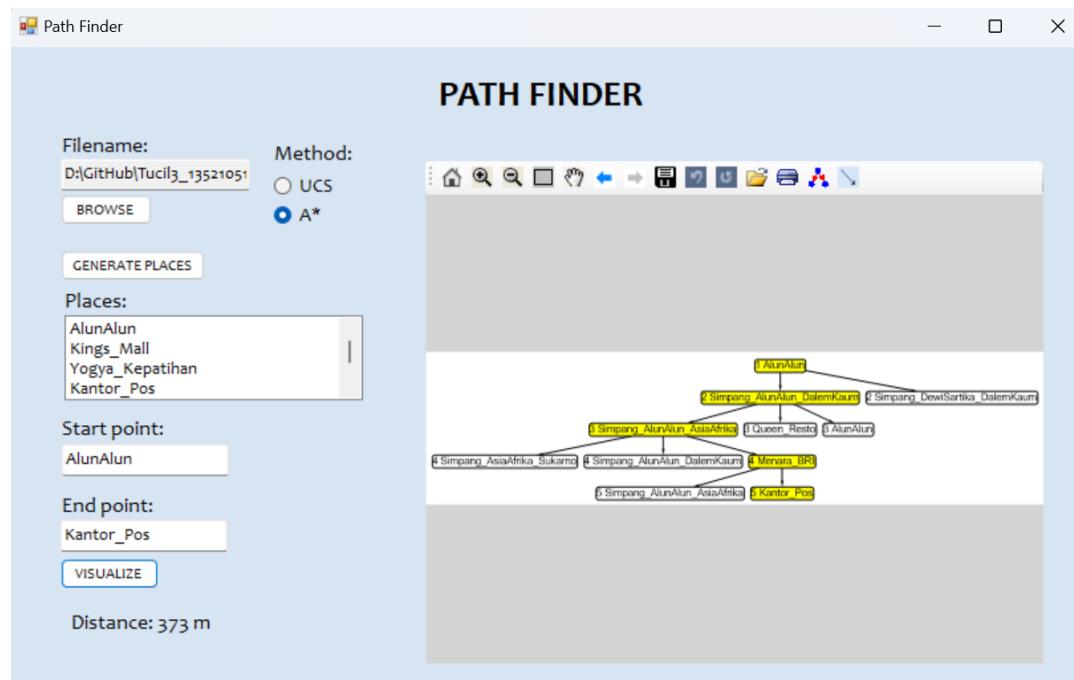
Pencarian:

Start : AlunAlun

End : Kantor_Pos



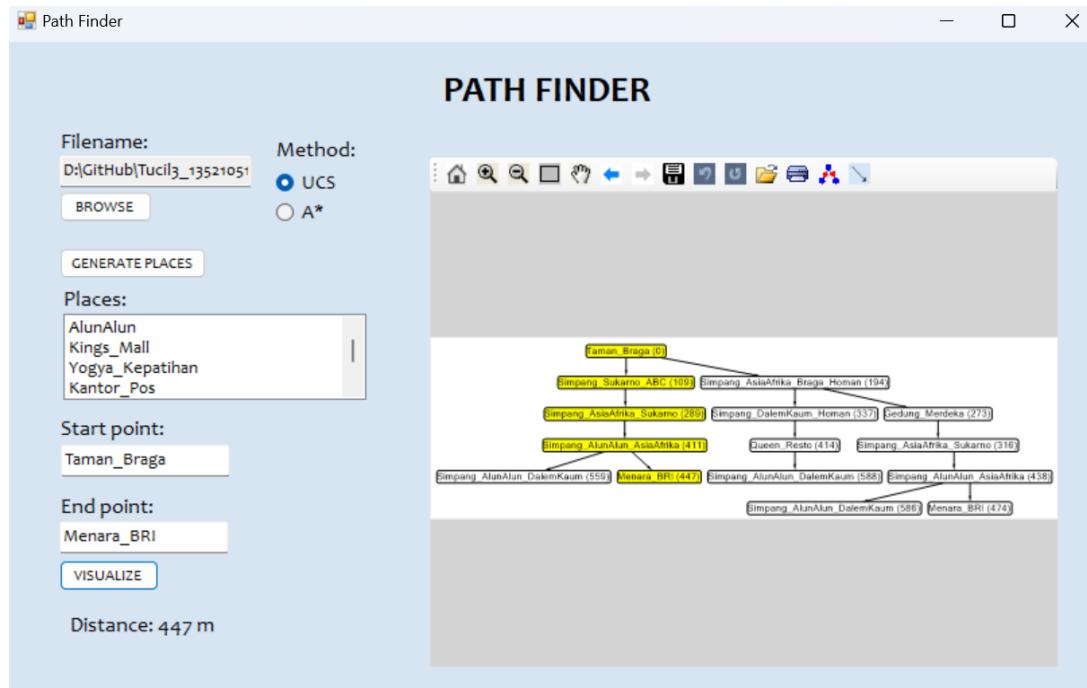
Gambar 3.2.3 Pengujian 2 dari Alun-Alun ke Kantor Pos (UCS)



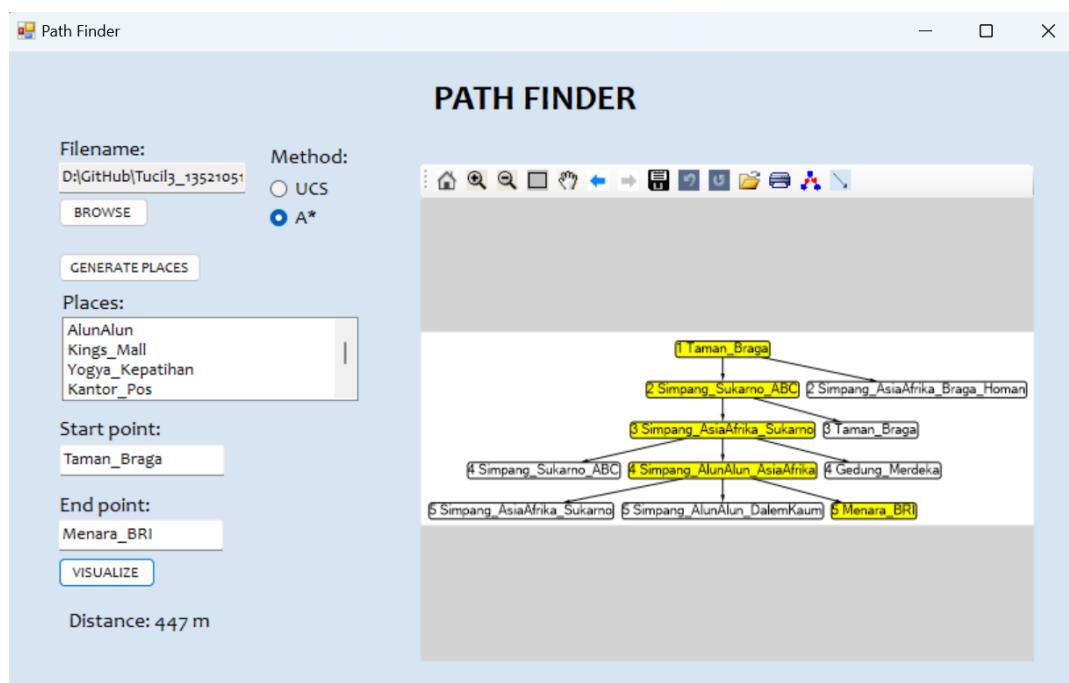
Gambar 3.2.4 Pengujian 2 dari Alun-Alun ke Kantor Pos (A*)

Start : Taman_Braga

End : Menara_BRI



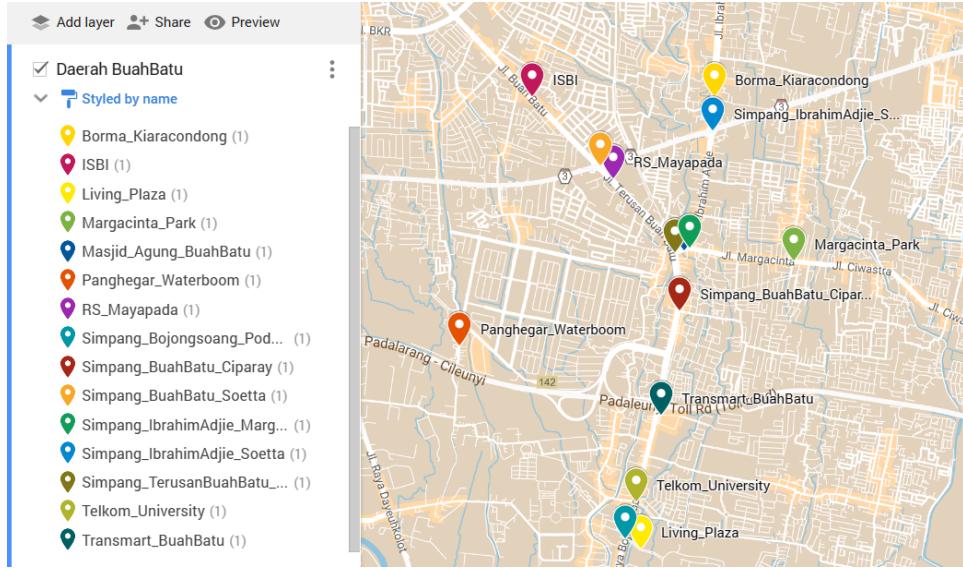
Gambar 3.2.5 Pengujian 2 dari Taman Braga ke Menara BRI (UCS)



Gambar 3.2.6 Pengujian 2 dari Taman Braga keMenara BRI (A*)

3.3 Pengujian 3: Peta Sekitar Buah Batu dan Bandung Selatan

Input peta: map3.txt



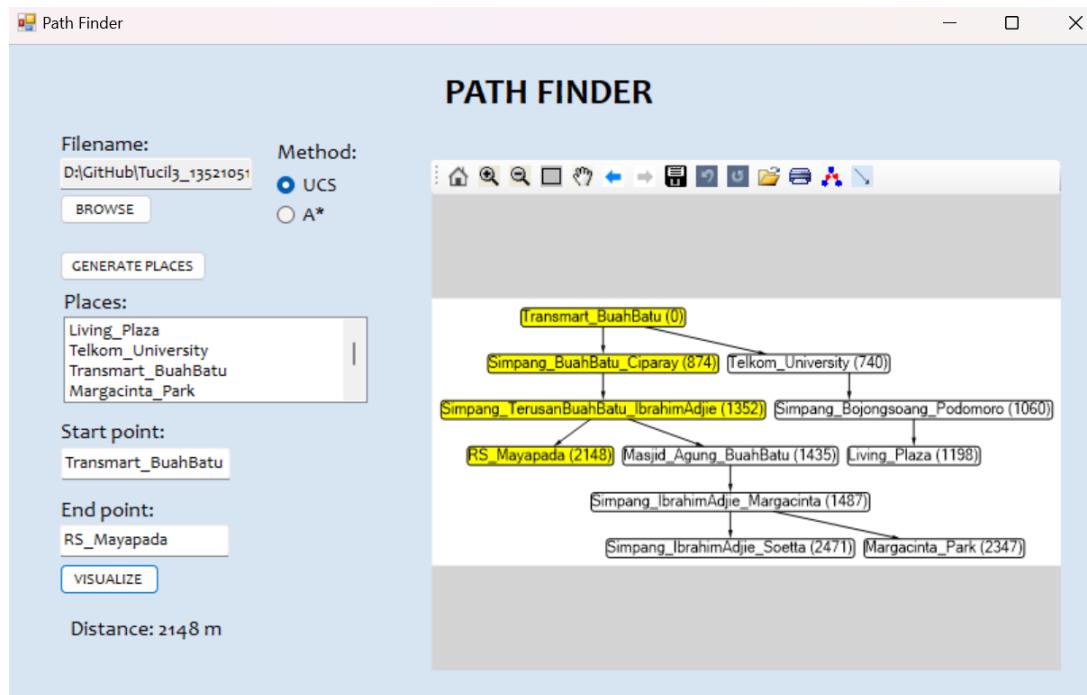
Gambar 3.3.1 Peta Sekitar Buah Batu dan Bandung Selatan

Gambar 3.3.2 Konfigurasi Graf Peta Sekitar Buah Batu dan Bandung Selatan (map3.txt)

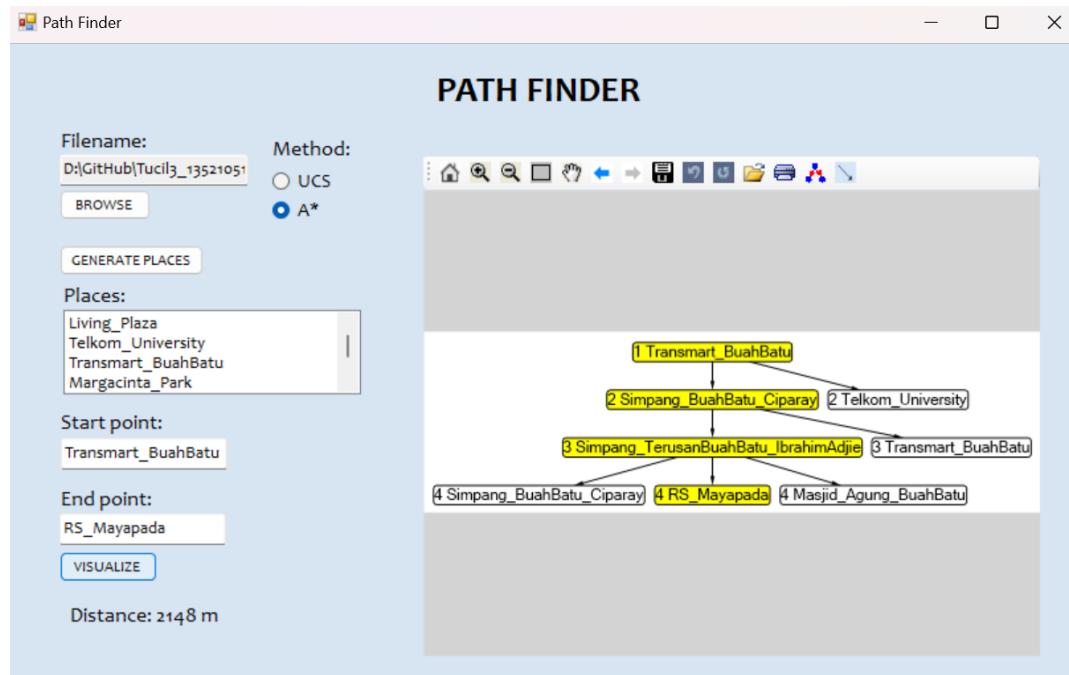
Pencarian:

Start : Transmart_BuahBatu

End : RS_Mayapada

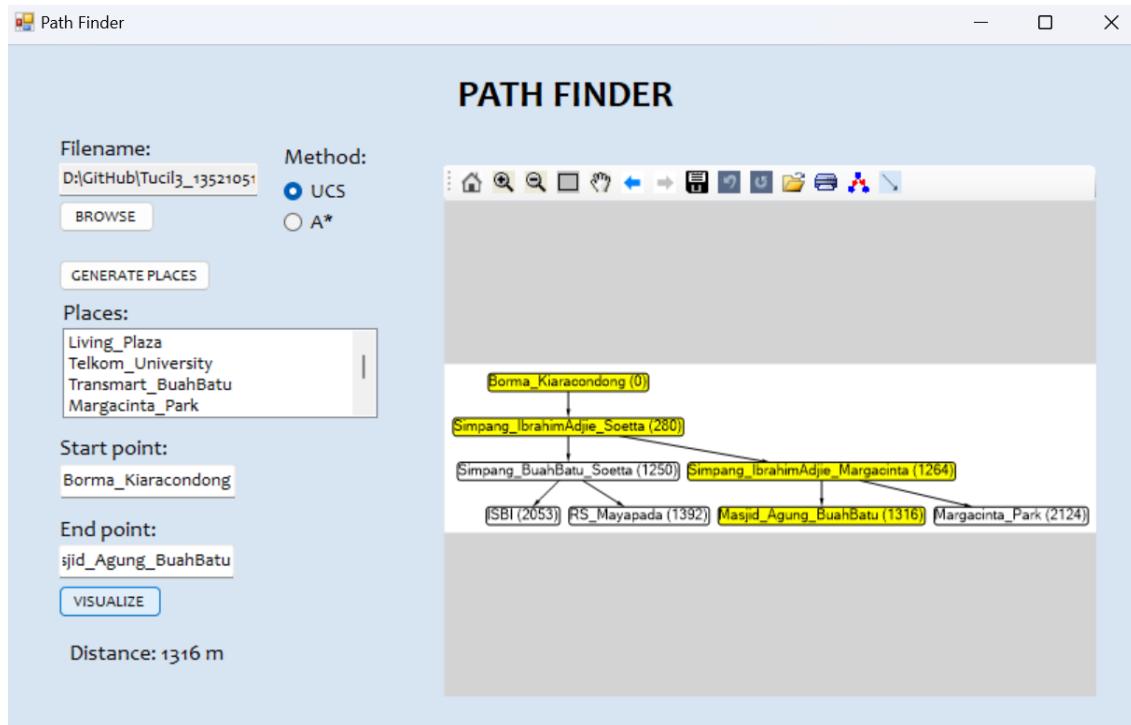


Gambar 3.3.3 Pengujian 3 dari Transmart Buah Batu ke Rumah Sakit Mayapada (UCS)

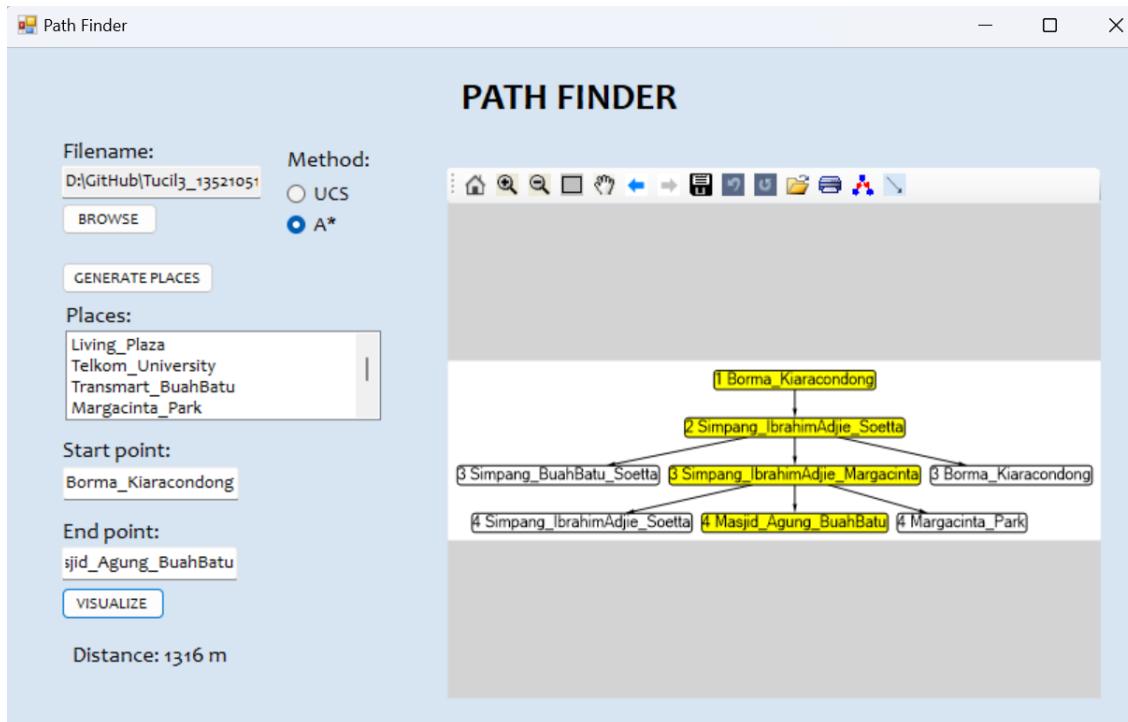


Gambar 3.3.4 Pengujian 3 dari Transmart Buah Batu ke Rumah Sakit Mayapada (A*)

Start : Borma_Kiaracondong
End : Masjid_Agung_BuahBatu



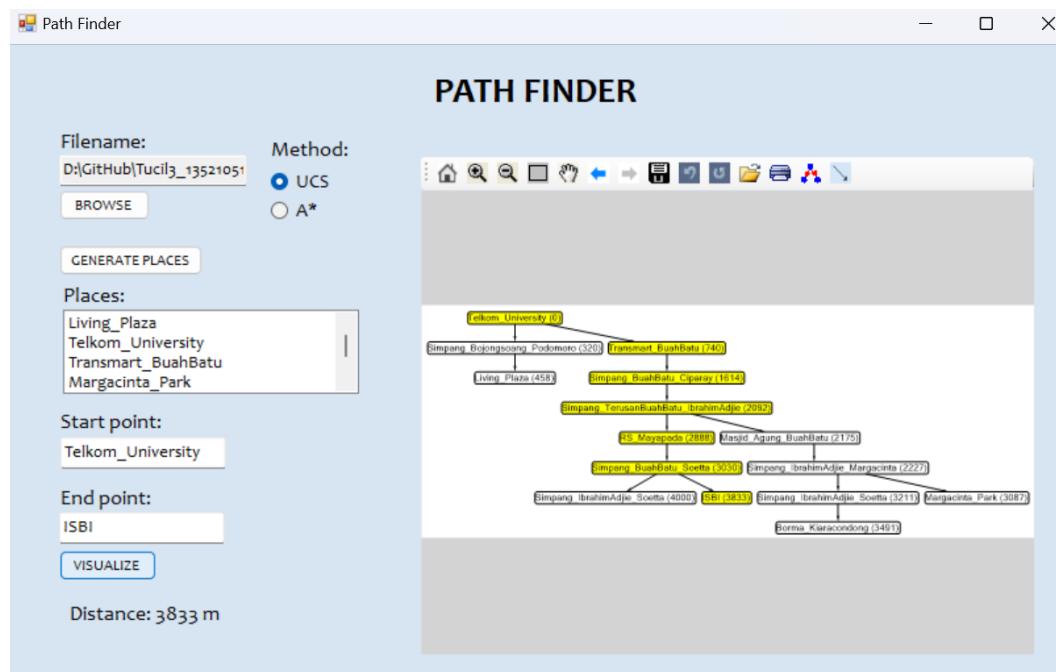
Gambar 3.3.5 Pengujian 3 dari Borma Kiaracondong ke Masjid Agung Buah Batu (UCS)



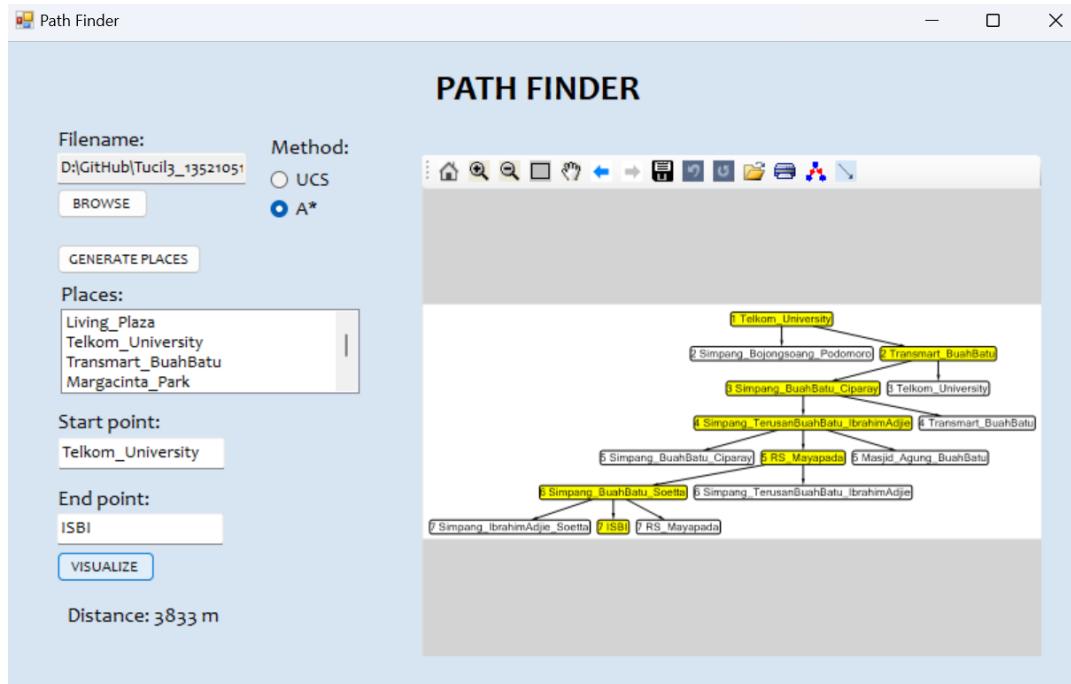
Gambar 3.3.6 Pengujian 3 dari Borma Kiaracondong ke Masjid Agung Buah Batu (A*)

Start : Telkom_University

End : ISBI



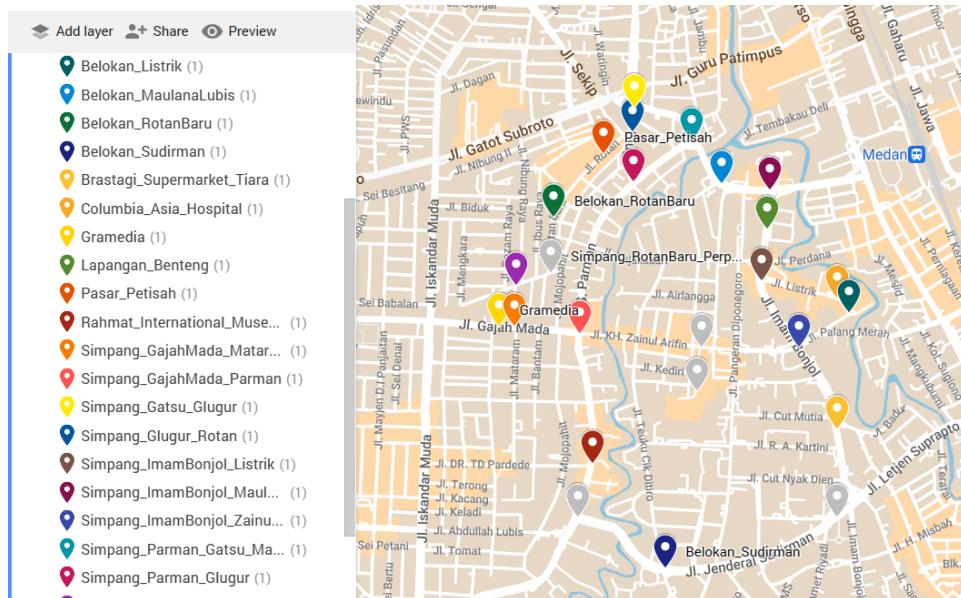
Gambar 3.3.7 Pengujian 3 dari Telkom University ke ISBI (UCS)



Gambar 3.3.8 Pengujian 3 dari Telkom University ke ISBI (A*)

3.4 Pengujian 4: Peta Daerah di Medan

Input peta: map4.txt



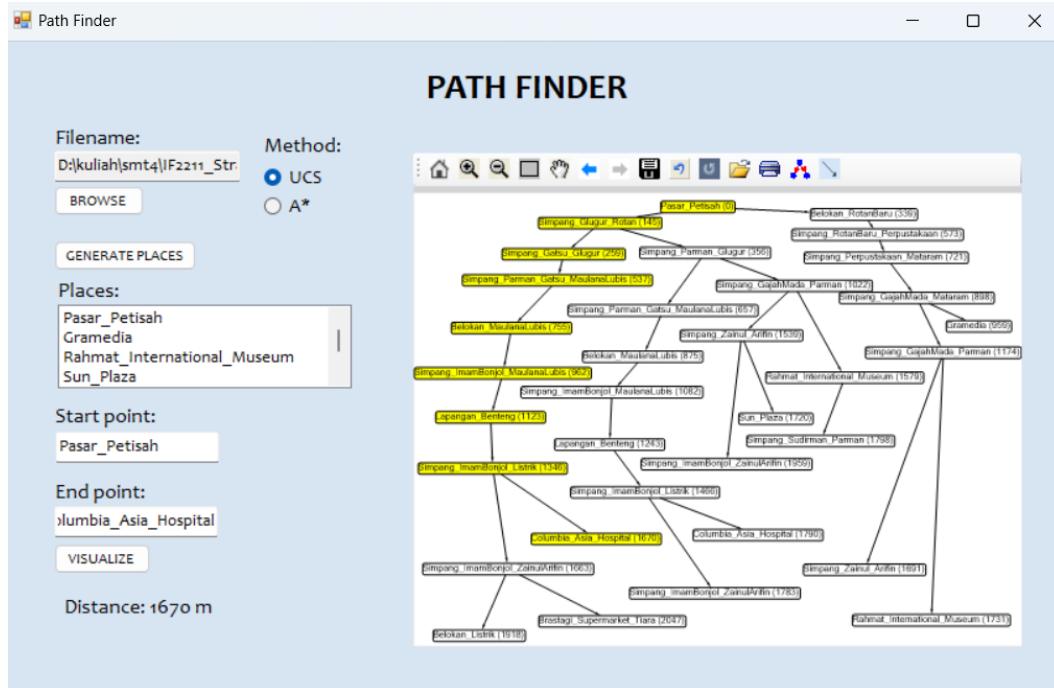
Gambar 3.4.1 Peta Daerah di Medan

Gambar 3.4.2 Konfigurasi Graf Peta Daerah di Medan (map4.txt)

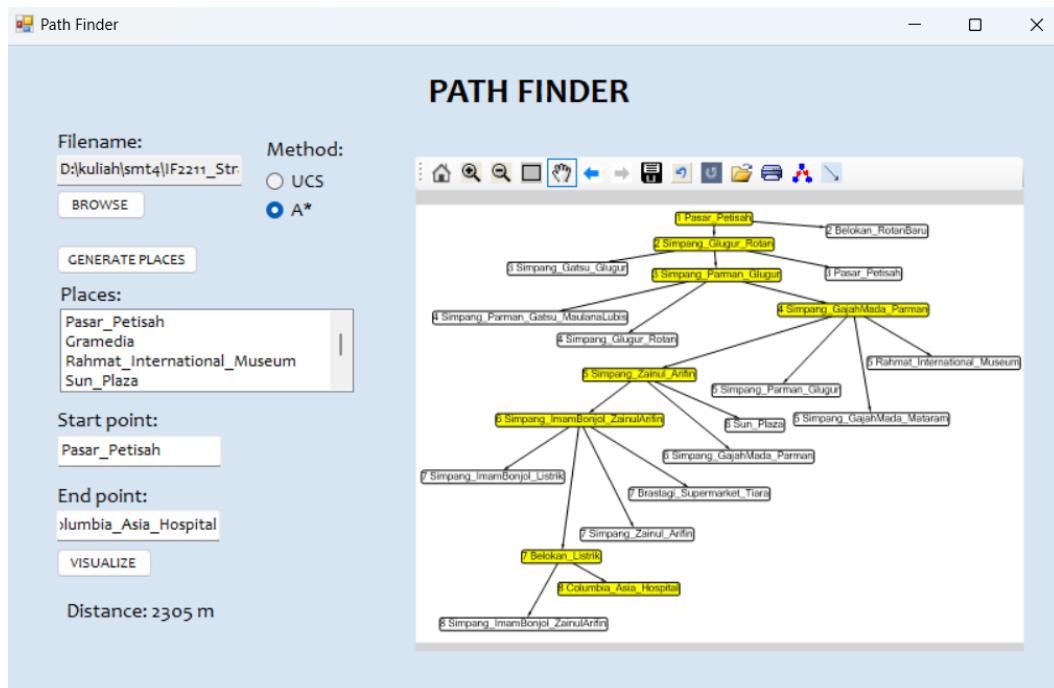
Pencarian

Start : Pasar Petisah

End : Columbia Asia Hospital

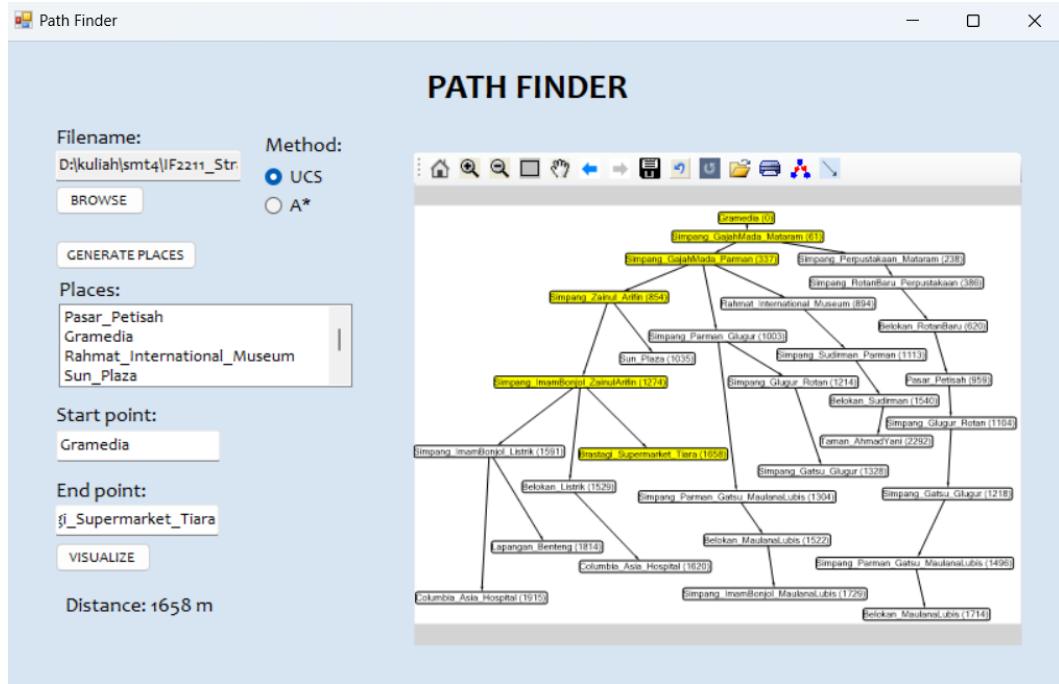


Gambar 3.4.3 Pengujian 4 dari Pasar Petisah ke Columbia Asia Hospital (UCS)

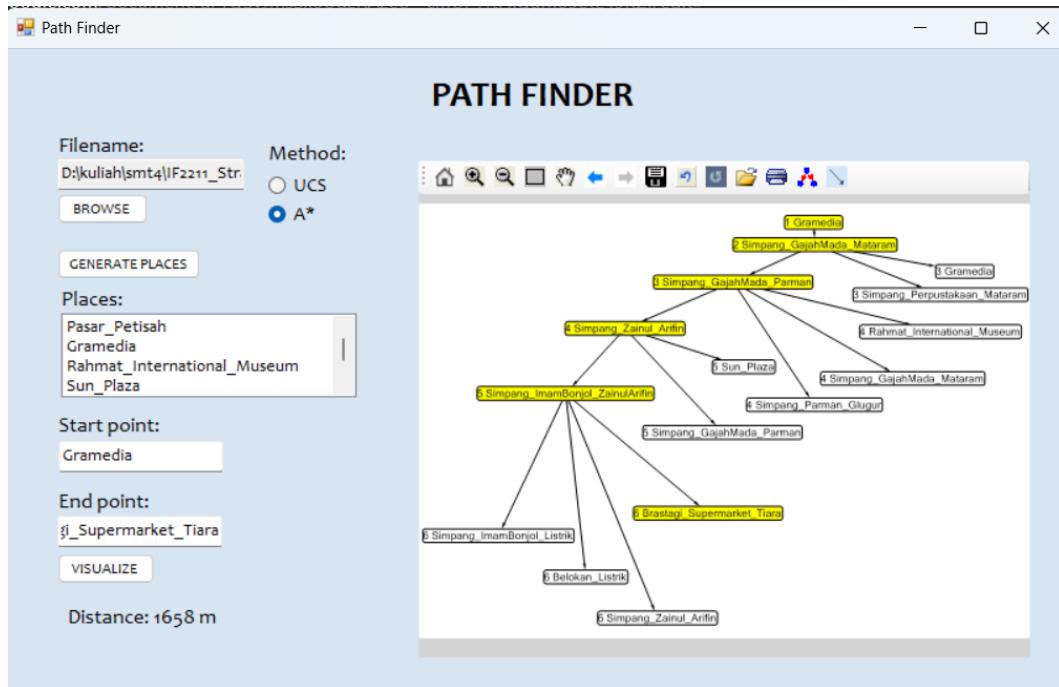


Gambar 3.4.4 Pengujian 4 dari Pasar Petisah ke Columbia Asia Hospital (A*)

Start : Gramedia
End : Brastagi_Supermarket_Tiara



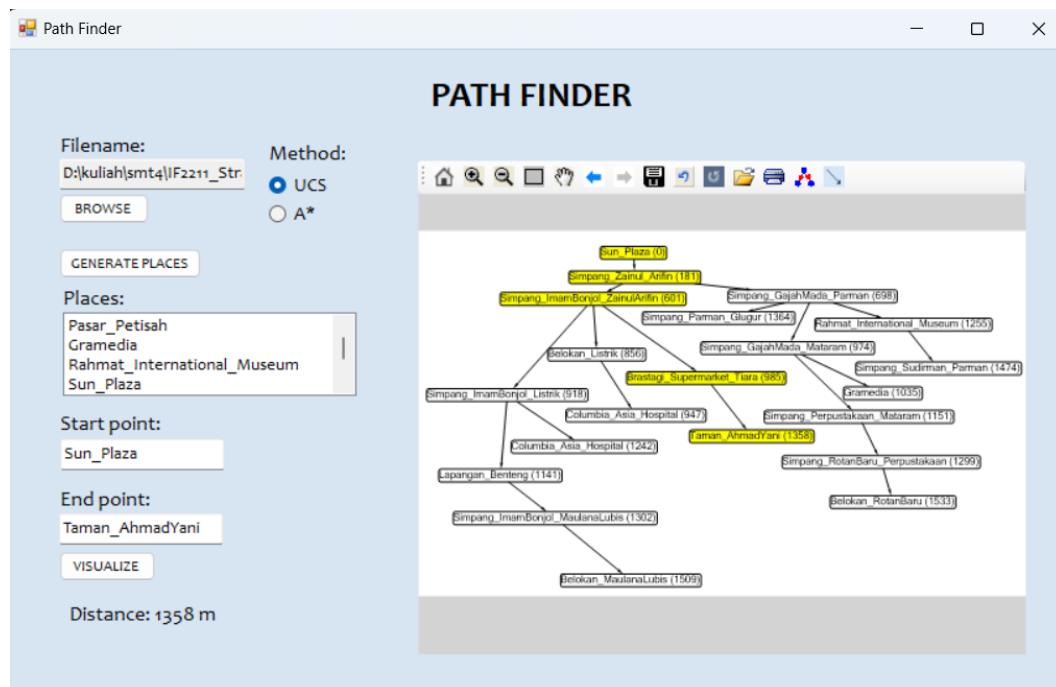
Gambar 3.4.5 Pengujian 4 dari Gramedia ke Brastagi Supermarket Tiara (UCS)



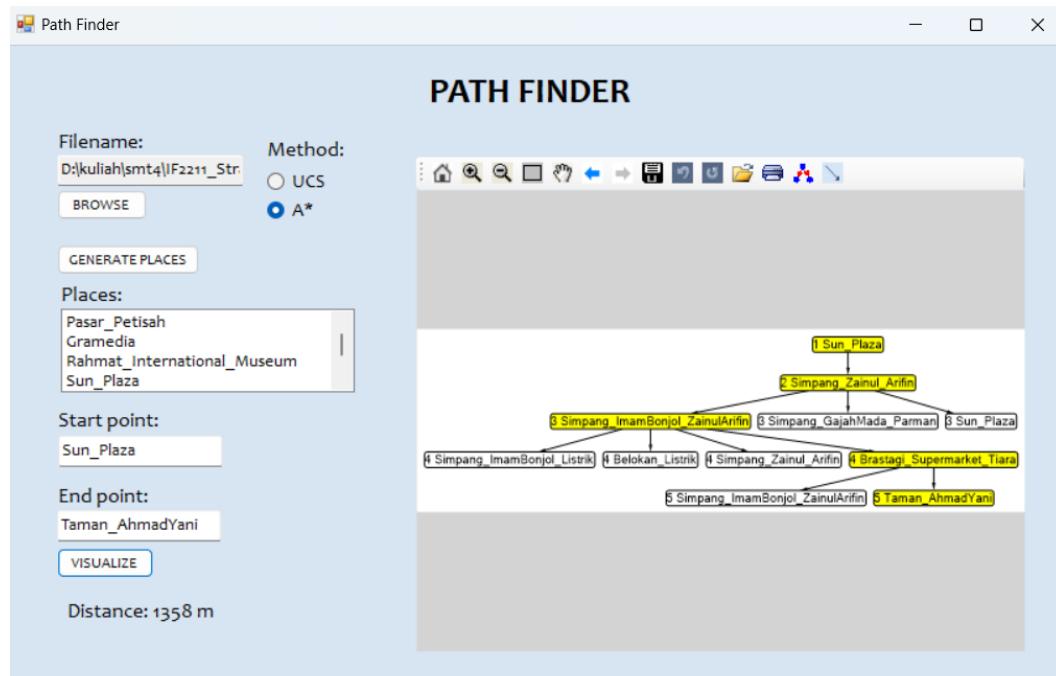
Gambar 3.4.6 Pengujian 4 dari Gramedia ke Brastagi Supermarket Tiara (A*)

Start : Sun_Plaza

End : Taman_AhmadYani



Gambar 3.4.7 Pengujian 4 dari Sun Plaza ke Taman Ahmad Yani (UCS)



Gambar 3.4.8 Pengujian 4 dari Sun Plaza ke Taman Ahmad Yani (A*)

3.5 Pengujian 5: Peta Sekitar Alun-Alun Bandung (Kasus Spesial)

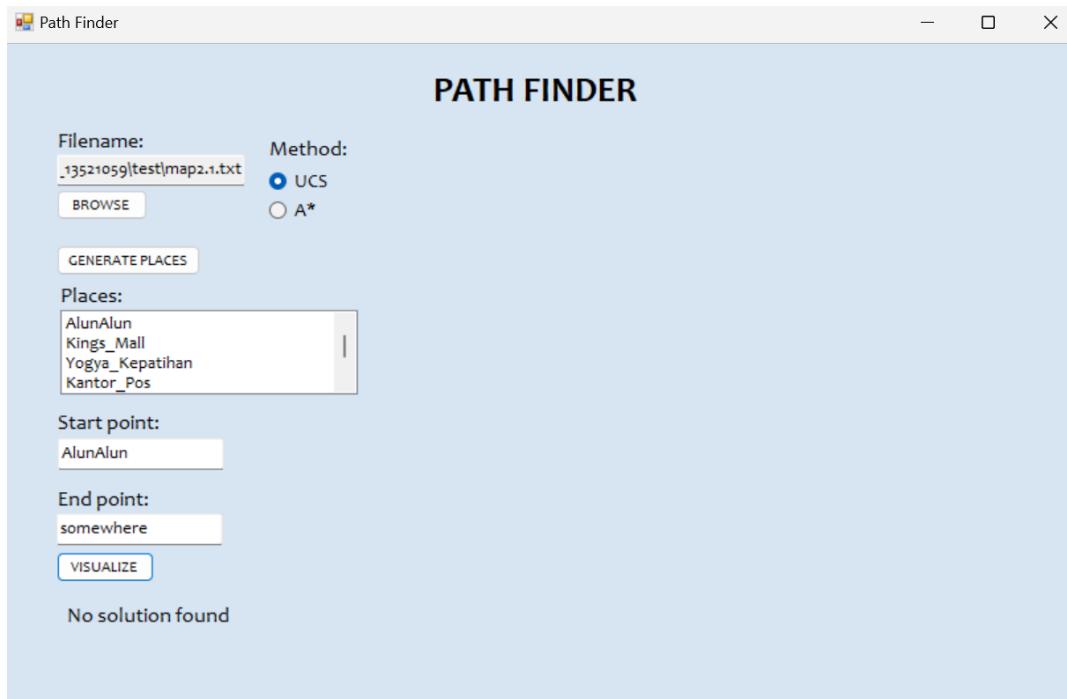
Input peta: map2.1.txt

Gambar 3.5.1 Konfigurasi Graf Peta Sekitar Alun-Alun Bandung - Kasus Spesial (map2.1.txt)

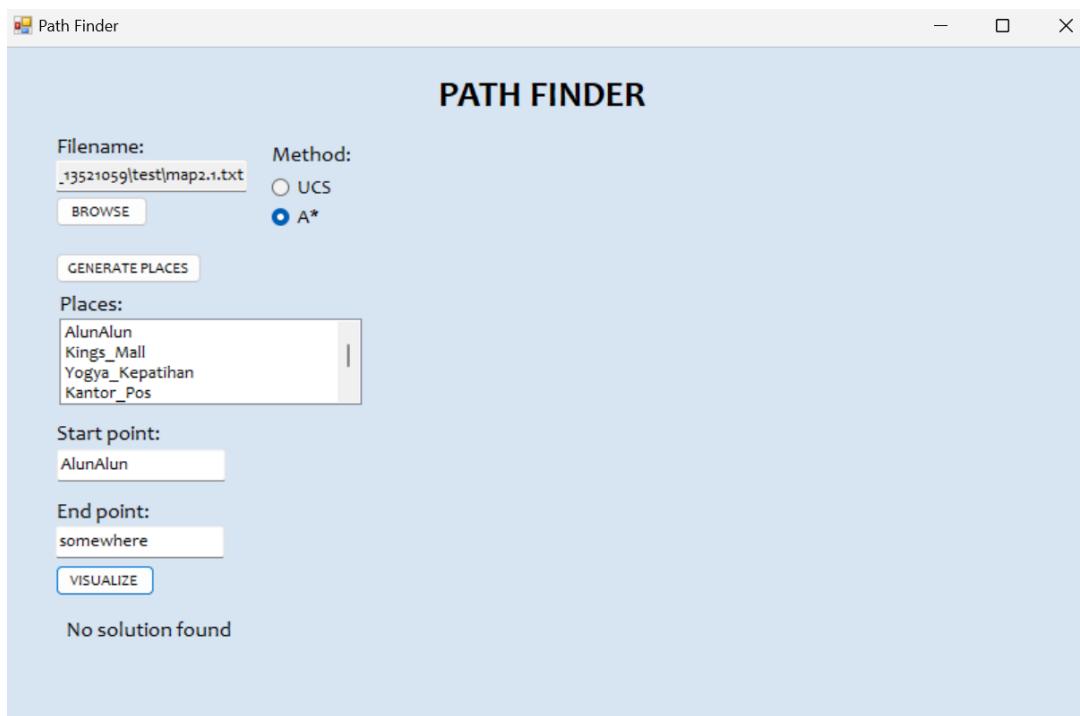
Pencarian:

Start : AlunAlun

End : somewhere



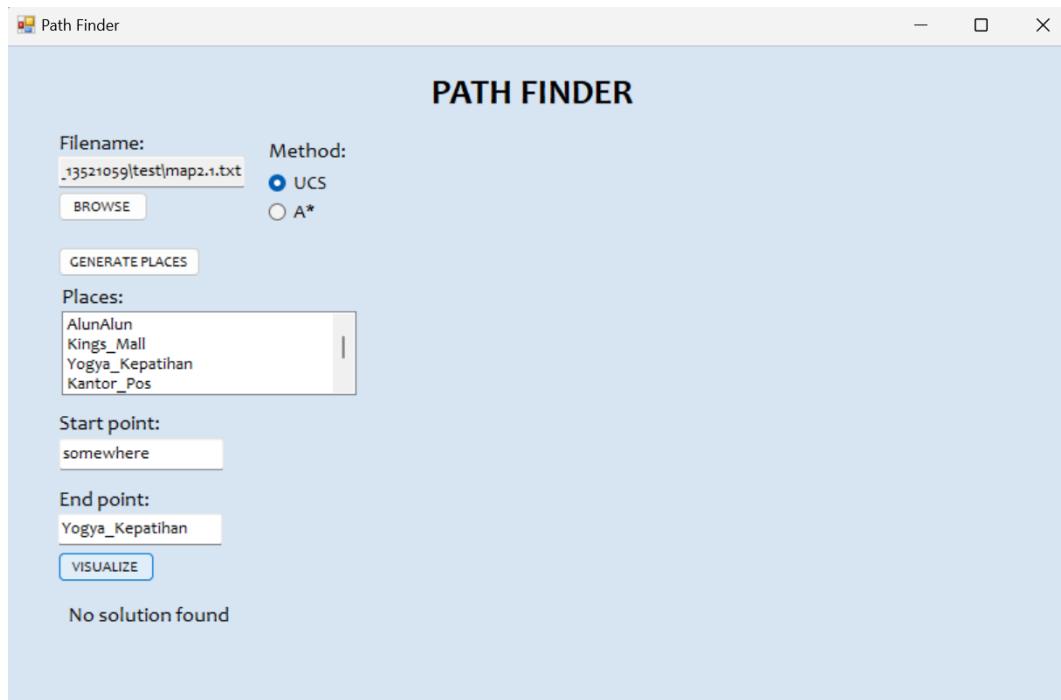
Gambar 3.5.2 Pengujian 5 dari Alun-Alun ke Somewhere (UCS)



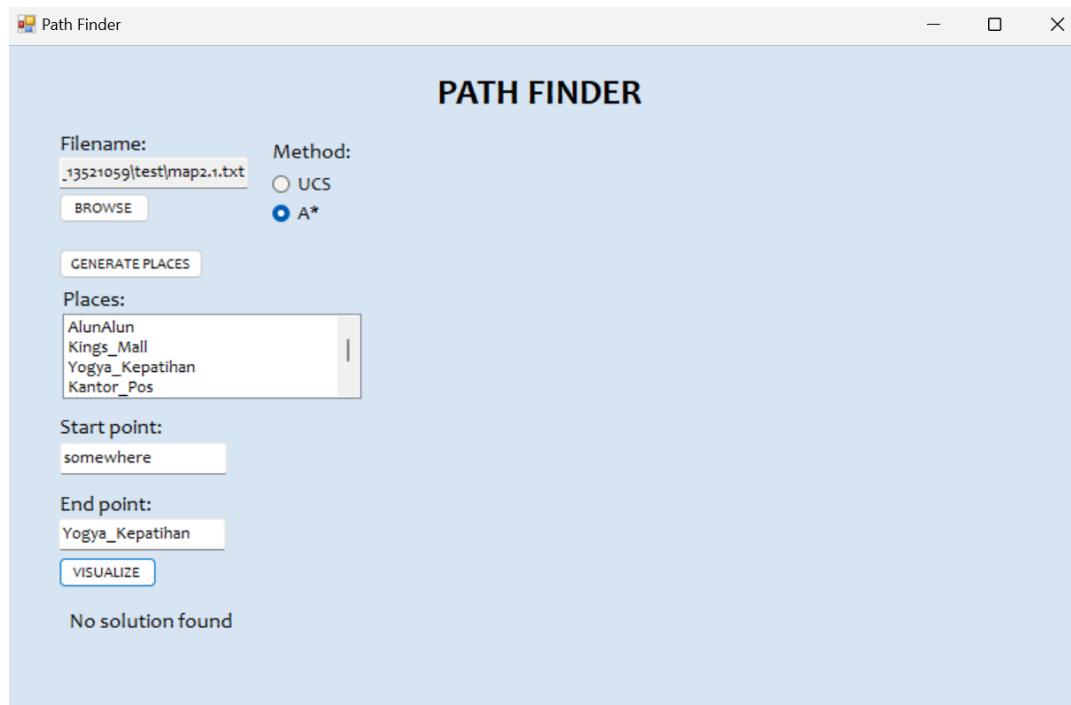
Gambar 3.5.3 Pengujian 5 dari Alun-Alun ke Somewhere (A*)

Start : somewhere

End : Yogyakarta_Kepatihan



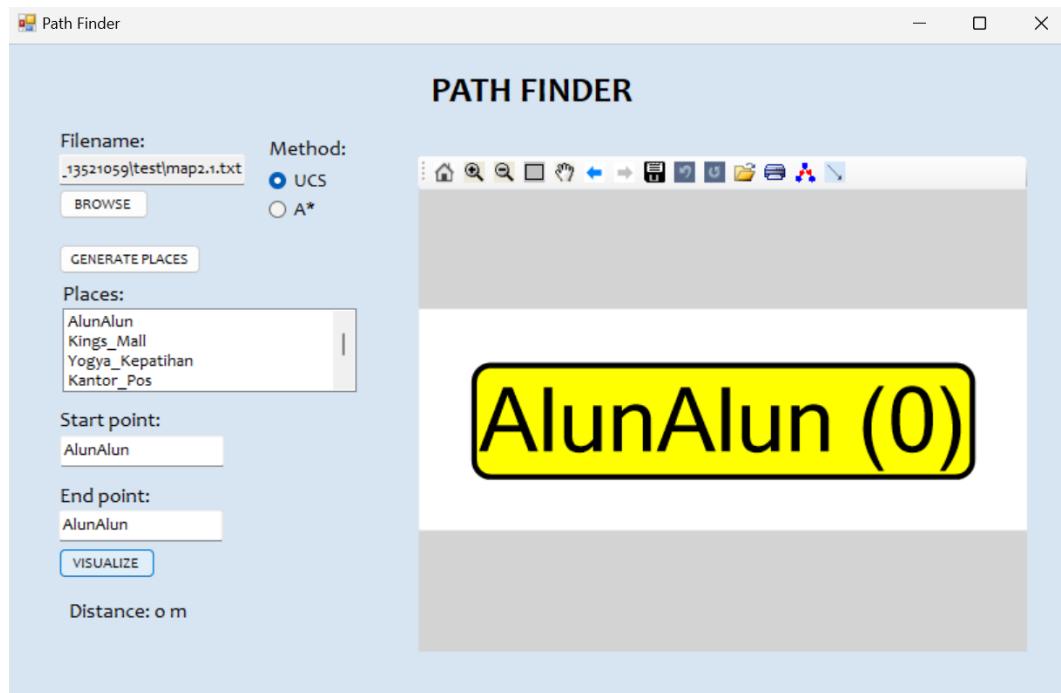
Gambar 3.5.4 Pengujian 5 dari Somewhere ke Yogyakarta Kepatihan (UCS)



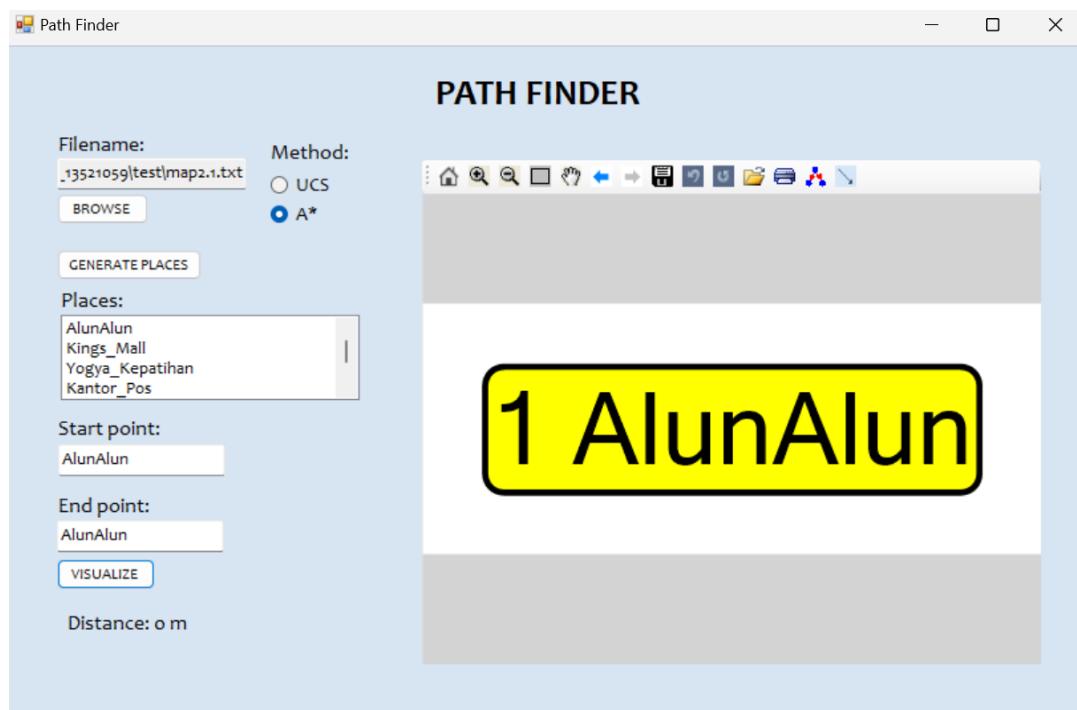
Gambar 3.5.5 Pengujian 5 dari Somewhere ke Yogyakarta Kepatihan (A*)

Start : AlunAlun

End : AlunAlun



Gambar 3.5.6 Pengujian 5 dari Alun-Alun ke Alun-Alun (UCS)



Gambar 3.5.7 Pengujian 5 dari Alun-Alun ke Alun-Alun(A*)

3.6 Pengujian 6: Peta Sekitar ITB (Kasus Spesial)

Misalkan ITB ke Simpang_Ganesa_Tamansari merupakan jalan satu arah (Simpang_Ganesa_Tamansari ke ITB tidak bisa dilalui), maka rute yang ditempuh akan berbeda apabila jalan tersebut dua arah.

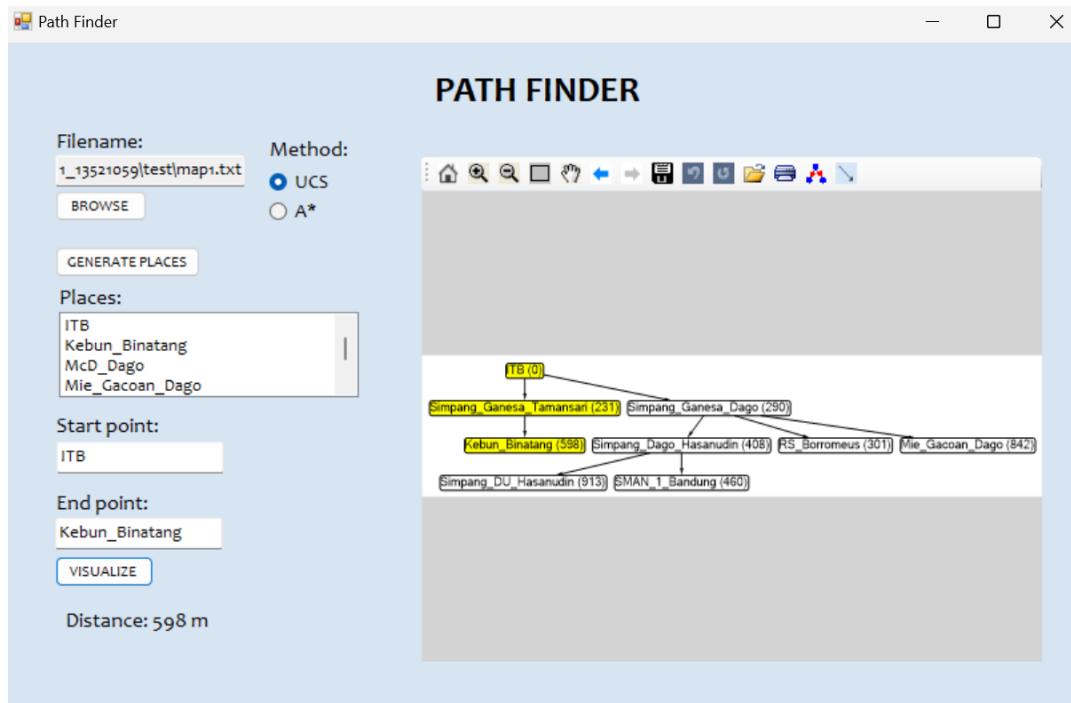
Input peta: map1.txt

Gambar 3.6.1 Konfigurasi Graf Peta Sekitar ITB (map1.txt)

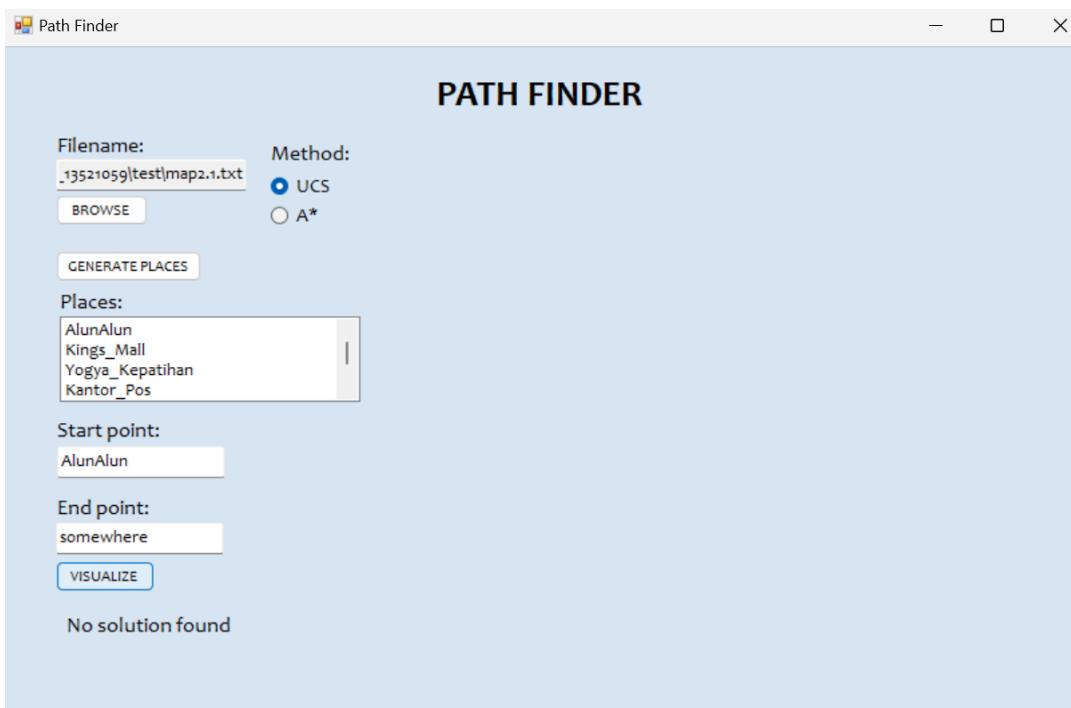
Pencarian:

Start : ITB

End : Kebun Binatang

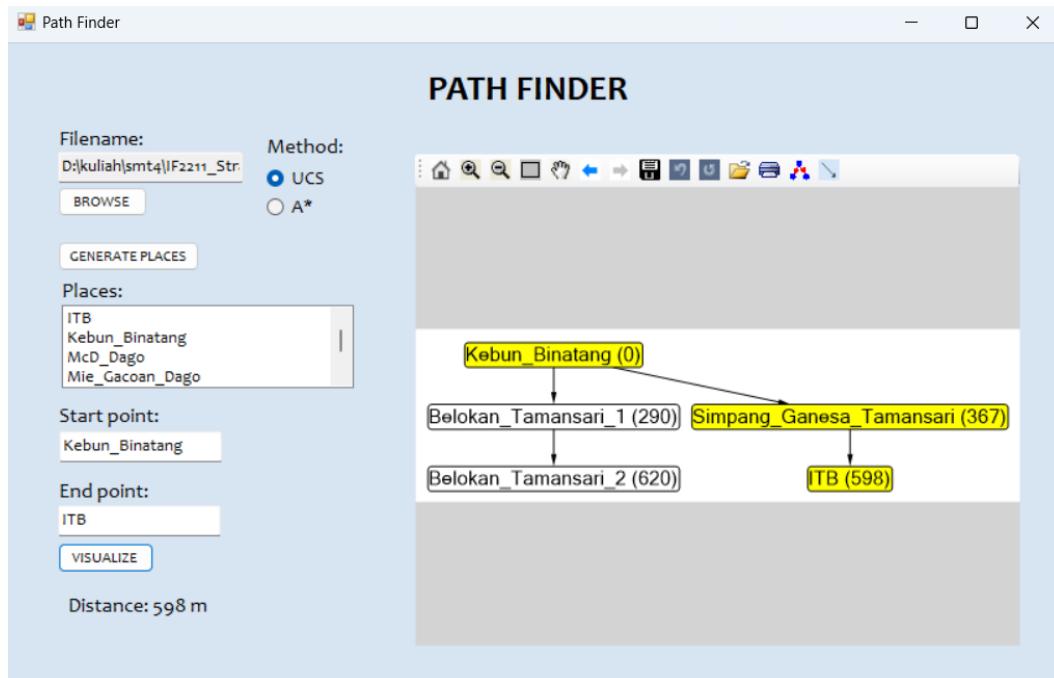


Gambar 3.6.2 Pengujian 6 dari ITB ke Kebun Binatang (UCS)

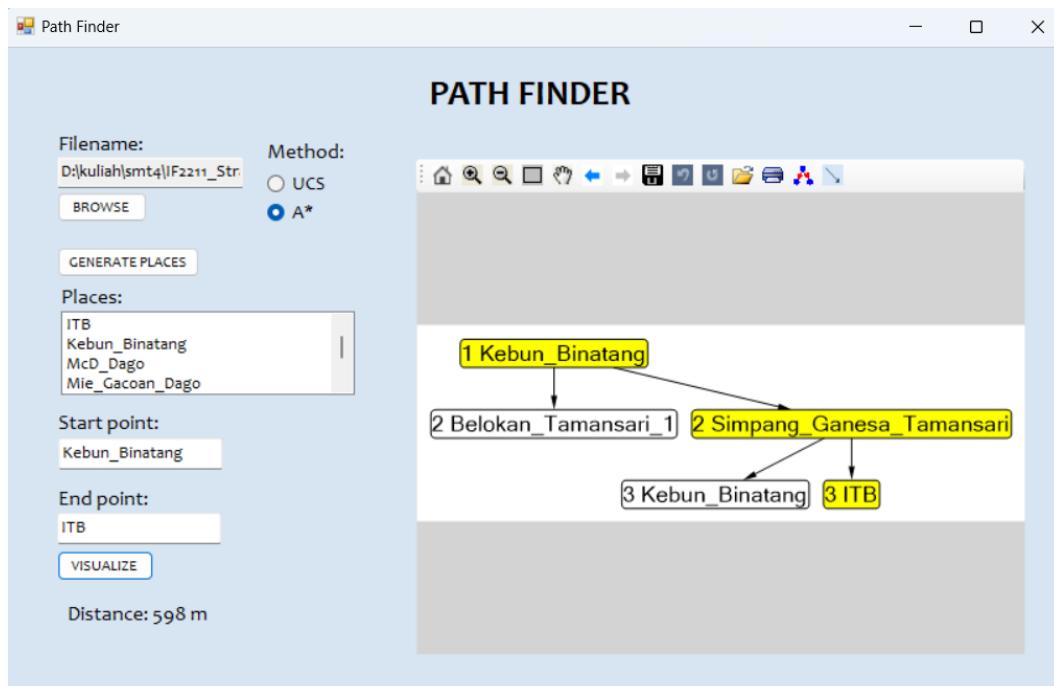


Gambar 3.6.3 Pengujian 6 dari Alun-Alun ke Somewhere (A*)

Start : Kebun_Binatang
End : ITB



Gambar 3.6.4 Pengujian 6 dari Kebun Binatang ke ITB (UCS)



Gambar 3.6.5 Pengujian 6 dari Kebun Binatang ke ITB (A*)

Kasus Khusus (Directed Graph):

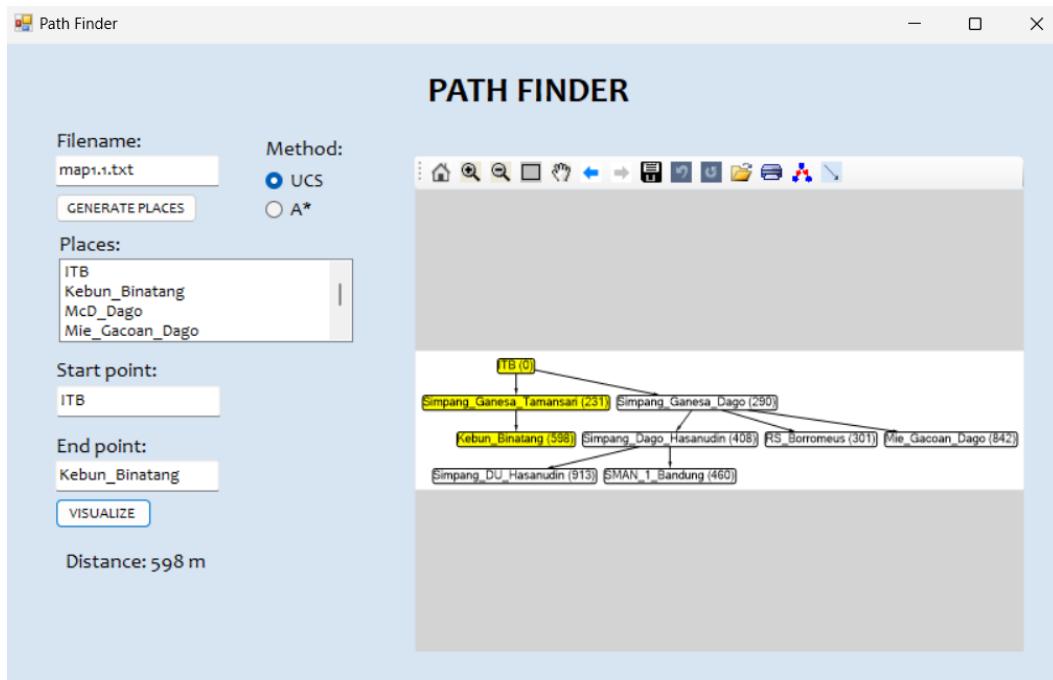
Input peta: map1.1.txt

Gambar 3.6.6 Konfigurasi Graf Peta Sekitar ITB Kasus Khusus (map1.1.txt)

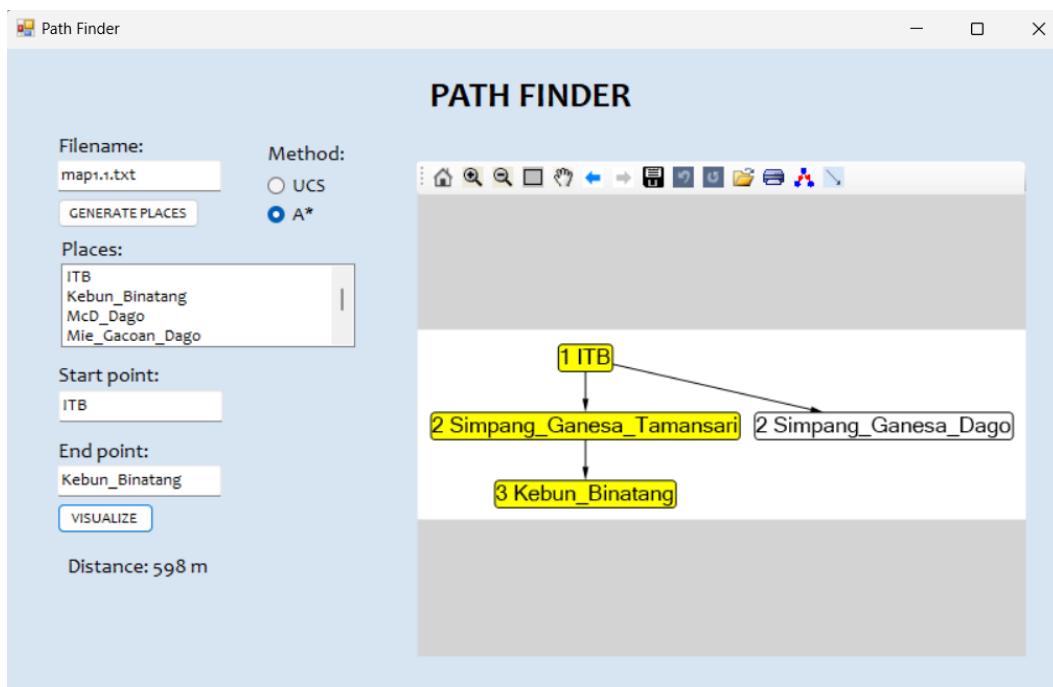
Pencarian:

Start : ITB

End : Kebun Binatang



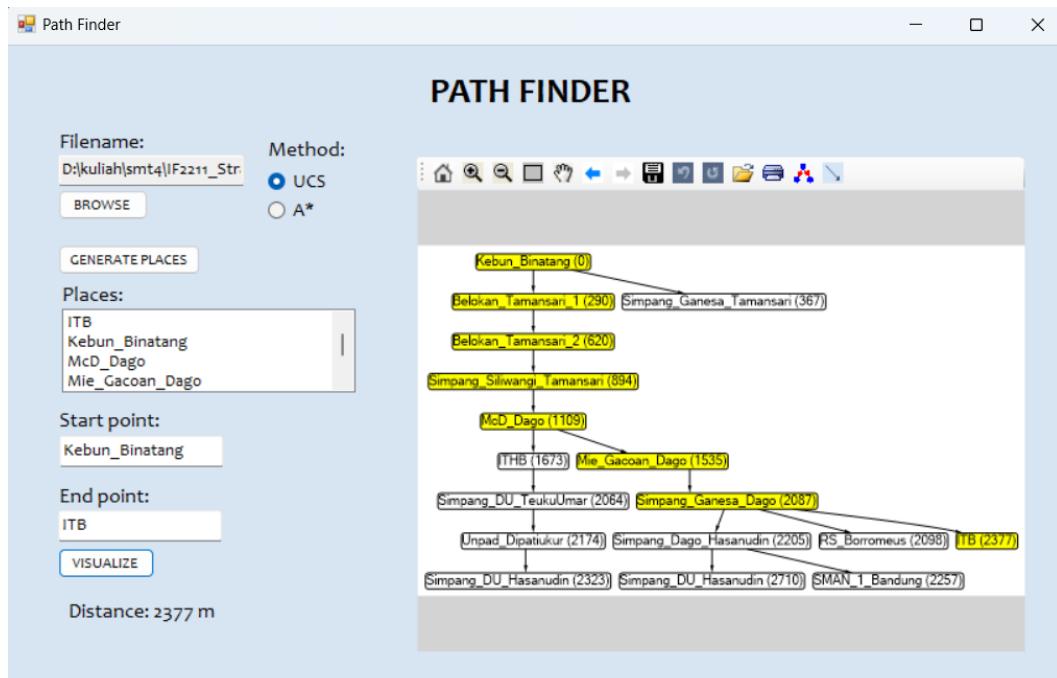
Gambar 3.6.7 Pengujian 6 dari ITB ke Kebun Binatang (UCS)



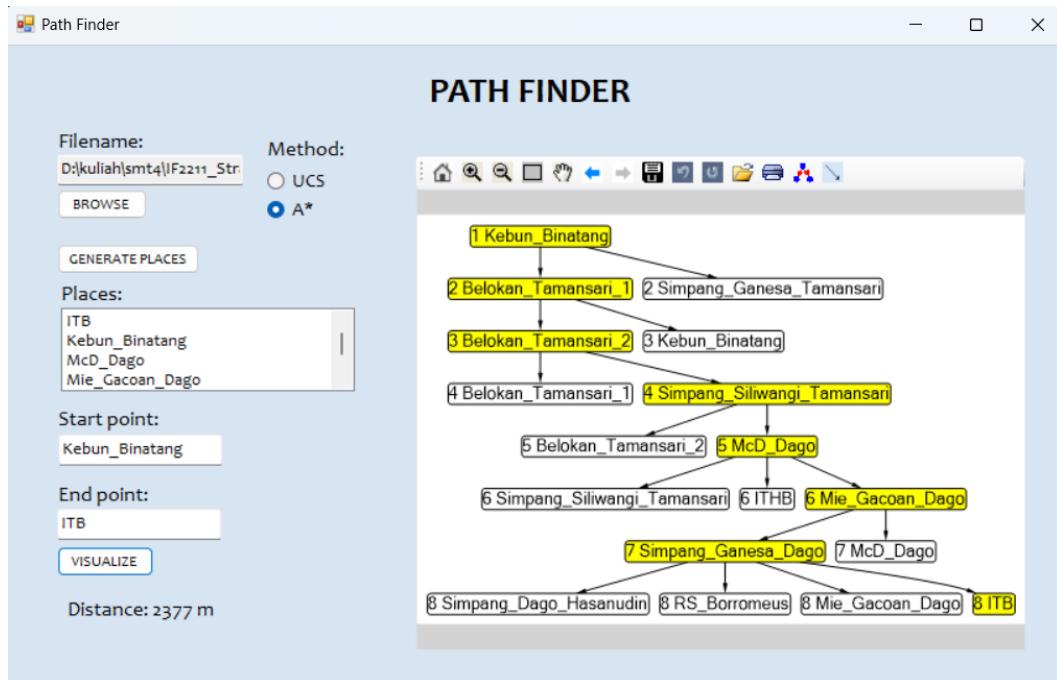
Gambar 3.6.8 Pengujian 6 dari Alun-Alun ke Somewhere (A*)

Start : Kebun_Binatang

End : ITB



Gambar 3.6.9 Pengujian 6 dari Kebun Binatang ke ITB (UCS)



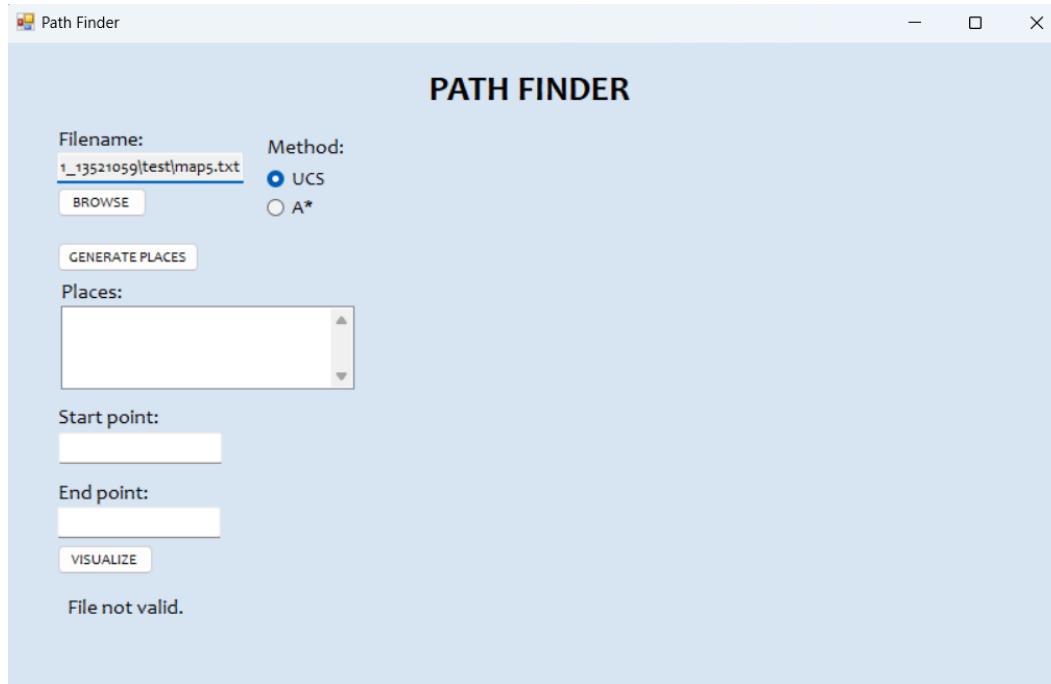
Gambar 3.6.10 Pengujian 6 dari Kebun Binatang ke ITB (A*)

3.7 Pengujian 7: File Tidak Valid

Misal terdapat konfigurasi peta yang tidak valid. Pada kasus ini, jumlah simpul yang tertulis pada baris pertama *file* lebih sedikit dibandingkan jumlah simpul yang didefinisikan pada baris-baris berikutnya.

Input peta: map2.1.txt

Gambar 3.7.1 Konfigurasi Graf Peta Tidak Valid (map5.txt)



Gambar 3.7.2 Pengujian 7 File Tidak Valid

IV. Kesimpulan dan Komentar

Melalui Tugas Kecil 3 ini, program pencarian rute terpendek dibuat. Algoritma yang dipakai untuk mencari rute terpendek dari suatu lokasi ke lokasi lainnya adalah UCS (Uniform Cost Search) dan A*. Program membaca graf peta dan lokasi awal dan tujuan, lalu melakukan pencarian rute terpendek dengan algoritma UCS atau A* serta menghitung jarak rute tersebut. Dengan demikian, dapat ditemukan rute terpendek antara dua lokasi pada peta. Algoritma UCS selalu menemukan rute terpendek, sedangkan A* tidak karena A* menggunakan pendekatan heuristik sehingga rute solusi belum tentu rute terpendek tapi membangkitkan lebih sedikit simpul.

V. Lampiran

4.1 Tautan Repotori GitHub

https://github.com/arleenchr/Tucil3_13521051_13521059

4.2 Tabel Penilaian

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	

VI. Referensi

Munir, Rinaldi. Sekolah Teknik Elektro dan Informatika ITB. 2023. “Penentuan rute (Route/Path Planning) - Bagian 1”.

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian_1-2021.pdf (Diakses 12 April 2023).

Munir, Rinaldi. Sekolah Teknik Elektro dan Informatika ITB. 2023. “Penentuan rute (Route/Path Planning) - Bagian 2”.

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian_2-2021.pdf (Diakses 12 April 2023).