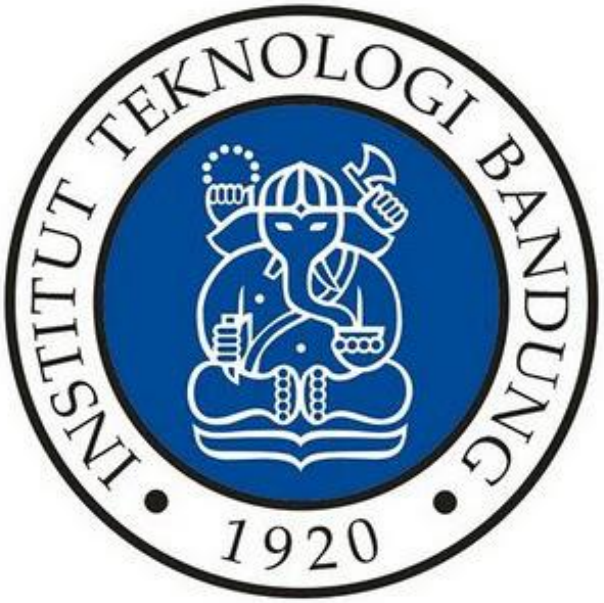
Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

13519181 - Nabilah Erfariani
13519192 - Gayuh Tri Rahutami



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

1. Source Code (C#)

1.1. Form1.cs

```
using System;
using System.Collections.Generic;
using Viewer = Microsoft.Msagl.GraphViewerGdi.GViewer;
using MsaglGraph = Microsoft.Msagl.Drawing.Graph;
using Form = System.Windows.Forms.Form;
using Color = Microsoft.Msagl.Drawing.Color;
using Drawing = Microsoft.Msagl.Drawing;
namespace PathFinder
{
    public partial class Form1 : Form
    {
        public string[] adjNode;
        public string[] identity;
        private Graph map;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
            string filename = openFileDialog1.FileName;
            //string readfile = File.ReadAllText(filename);
            textBox1.Text = filename;

            map = new Graph(filename);

            List<string> a = map.GetNodeNames();
            List<string> b = map.GetNodeNames();
            comboBox1.DataSource = a;
            comboBox2.DataSource = b;

            //create a viewer object
            Viewer viewer = new Viewer();
            //create a graph object
            MsaglGraph graphh = new MsaglGraph("graphh");
            //create the graph content

            foreach (Node node in map.GetNodes())
            {
                foreach (int adjNode in node.GetAdjList())
                {
                    if (node.GetID() < adjNode) {
                        var Edge = graphh.AddEdge(node.GetName(),
                                    map.GetNode(adjNode).GetName());
                        Edge.Attr.ArrowheadAtTarget = Drawing.ArrowStyle.None;
                        Edge.LabelText = node.CalculateDistance (map.GetNode
                                        (adjNode)).ToString() + " m";
                    }
                }
            }
```

```
        viewer.Graph = graphh;
        //associate the viewer with the form
        panel1.SuspendLayout();
        viewer.Dock = System.Windows.Forms.DockStyle.Fill;
        if (panel1.Controls.Count != 0) panel1.Controls.RemoveAt(0);
        panel1.Controls.Add(viewer);
        panel1.ResumeLayout();

    }

    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        FindPath();
    }

    private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
    {
        FindPath();
    }

    private void FindPath(){
        if(comboBox1.Text == "" || comboBox2.Text == "") return;

        PathFinder findPath = new PathFinder(comboBox1.Text,
                                  comboBox2.Text, map);

        List<Node> path = findPath.GetPath();
        //create a viewer object
        Viewer viewer = new Viewer();
        //create a graph object
        MsaglGraph graphhh = new MsaglGraph("graphhh");
        //create the graph content

        List<string> highlightedEdges = new List<string>();
        for(int i = 0; i < path.Count - 1; i++)
        {
            var Edge = graphhh.AddEdge(path[i].GetName(),
                       path[i+1].GetName());
            highlightedEdges.Add(path[i].GetID() + " " +
                                 path[i+1].GetID());
            highlightedEdges.Add(path[i+1].GetID() + " " +
                                 path[i].GetID());
            Edge.Attr.Color = Color.Coral;
            Edge.Attr.ArrowheadAtTarget = Drawing.ArrowStyle.None;
            Edge.LabelText = path[i].CalculateDistance
                            (path[i+1]).ToString() + " m";
        }

        foreach (Node node in map.GetNodes())
        {
            foreach (int adjNode in node.GetAdjList())
            {
                if (node.GetID() < adjNode && !highlightedEdges.
                    Contains(node.GetID() + " " + adjNode)) {
                    var Edge = graphhh.AddEdge(node.GetName(),
                               map.GetNode(adjNode).GetName());
                    Edge.Attr.ArrowheadAtTarget = Drawing.ArrowStyle.None;
                    Edge.LabelText = node.CalculateDistance(map.GetNode
```

```
                                        (adjNode)).ToString() + " m";
                    }
                }
            }

            foreach (Drawing.Node node in graphhh.Nodes)
            {
                node.Attr.Color = Color.LightBlue;
            }

            foreach (Node node in path){
                graphhh.FindNode(node.GetName()).Attr.FillColor =
                Color.Yellow;
            }

            viewer.Graph = graphhh;
            //associate the viewer with the form
            panel1.SuspendLayout();
            viewer.Dock = System.Windows.Forms.DockStyle.Fill;
            if(panel1.Controls.Count != 0) panel1.Controls.RemoveAt(0);
            panel1.Controls.Add(viewer);
            panel1.ResumeLayout();

            richTextBox5.Text = findPath.GetDistance().ToString() + " m";
        }
    }
}
```

1.2. PathFinder.cs

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

namespace PathFinder
{
    class PathFinder
    {
        List<Node> path;
        double distance;

        public PathFinder(string start, string destination, Graph map)
        {
            Node startNode = map.GetNode(start);
            Node destNode = map.GetNode(destination);
            Node currentNode = startNode;

            if (startNode == null || destNode == null)
            {
                path = new List<Node>();
                distance = -1;
                return;
            }

            SortedQueue queue = new SortedQueue();

            currentNode.SetDistanceFromStart(0);
```

```
            currentNode.SetEstimatedDistance
            (currentNode.CalculateDistance(destNode));

            currentNode.Visit();
            while (currentNode != null)
            {
                currentNode.Visit();
                foreach (int adjID in currentNode.GetAdjList())
                {
                    Node adjNode = map.GetNode(adjID);

                    if (adjNode.CalculateDistance(currentNode) +
                        adjNode.CalculateDistance(destNode) +
                        currentNode.GetDistanceFromStart() <
                        adjNode.GetEstimatedDistance() ||
                        adjNode.GetEstimatedDistance() == -1)
                    {
                        adjNode.SetParentID(currentNode.GetID());

                        adjNode.SetDistanceFromStart
                        (adjNode.CalculateDistance(currentNode) +
                        currentNode.GetDistanceFromStart());

                        adjNode.SetEstimatedDistance
                        (adjNode.GetDistanceFromStart() +
                        adjNode.CalculateDistance(destNode));
                    }

// Kalo node belom divisit dan node bukan destination maka
dimasukkin ke queue
                    if (!adjNode.GetVisited() && !Equals(adjNode, destNode))
                    {
                        queue.Enqueue(adjNode);
                    }
                }
                currentNode = queue.Dequeue();
            }
            CreatePath(startNode, destNode, map);
            distance = destNode.GetDistanceFromStart();
            map.Clear();
        }

        public void CreatePath(Node start, Node destination, Graph map)
        {
            Node currentNode = destination;
            path = new List<Node>();

            if (destination.GetParentID() != 0)
            {
                do
                {
                    path.Add(currentNode);
                    currentNode = map.GetNode(currentNode.GetParentID());
                } while (!Equals(start, currentNode));
                path.Add(currentNode);
                path.Reverse();
            }
        }
```

```
        public List<Node> GetPath()
        {
            return path;
        }

        public double GetDistance()
        {
            return distance;
        }
    }
}
```

1.3. Graph.cs

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

namespace PathFinder
{
    class Graph
    {
        private List<Node> nodes;

        // Constructor
        public Graph(string path)
        {
            nodes = new List<Node>();

            StreamReader graphFile = new StreamReader(path);
            int n;

            Int32.TryParse(graphFile.ReadLine(), out n);

            for (int i = 1; i <= n; i++)
            {
                double latitude, longitude;
                string name = "";

                // format per line latitude longitude Nama
                string[] identity = graphFile.ReadLine().Split(" ");
                latitude = Double.Parse(identity[0],
                           System.Globalization.CultureInfo.InvariantCulture);
                longitude = Double.Parse(identity[1],
                           System.Globalization.CultureInfo.InvariantCulture);

                // Double.TryParse(identity[0], out latitude);
                // Double.TryParse(identity[1], out longitude);
                List<string> list = new List<string>();
```

```
            for (int j = 2; j < identity.Length; j++)
            {
                name += identity[j];
                list.Add(name);
                string[] str = list.ToArray();

                if (j != identity.Length - 1) name += " ";
            }

            Node node = new Node(name, i, latitude, longitude);
            InsertNode(node);
        }

        for (int i = 0; i < n; i++)
        {
            string[] adjNode = graphFile.ReadLine().Split(" ");
            for (int j = 0; j < n; j++)
            {
                if (adjNode[j] == "1")
                {
                    GetNode(i + 1).insertAdjNode(j + 1);
                }
            }
        }

        graphFile.Close();
    }

    // Mengembalikan list of nama-nama node yang ada di graf
    public List<string> GetNodeNames()
    {
        List<string> names = new List<string>();

        foreach (Node node in nodes)
        {
            names.Add(node.GetName());
        }

        return names;
    }

    // Menambahkan node ke dalam graf
    public void InsertNode(Node node)
    {
        nodes.Add(node);
    }

    // Mengembalikan node yang memiliki ID id
    public Node GetNode(int id)
    {
        foreach(Node node in nodes)
        {
            if (node.GetID() == id) return node;
```

```
            }
            return null;
        }

        // Mengembalikan node yang memiliki nama name
        public Node GetNode(string name)
        {
            foreach (Node node in nodes)
            {
                if (node.GetName().ToLower() == name.ToLower()) return node;
            }
            return null;
        }

        // Mengambil semua node yang ada di graf
        public List<Node> GetNodes()
        {
            return nodes;
        }

// Mereset distance dan status visited
        public void Clear()
        {
            foreach(Node node in nodes)
            {
                node.SetDistanceFromStart(-1);
                node.SetEstimatedDistance(-1);
                node.SetParentID(0);
                node.UnVisit();
            }
        }
    }
}
```

1.4. Node.cs

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

namespace PathFinder
{
    class Node
    {
        private string name;
        private int id;
        private double latitude;
        private double longitude;
        private List<int> adjList;
        private double estimatedDistance;
        private double distanceFromStart;
        private int parentID;
        private bool visited;
```

```
        public Node(string name, int id, double latitude, double longitude)
        {
            this.name = name;
            this.id = id;
            this.latitude = latitude;
            this.longitude = longitude;
            adjList = new List<int>();
            estimatedDistance = -1;
            distanceFromStart = -1;
            parentID = 0;
            visited = false;
        }

        public void insertAdjNode(int adjNodeID)
        {
            adjList.Add(adjNodeID);
        }

        // Menghitung jarak antara this dan node dengan menggunakan haversine
function
        public double CalculateDistance(Node node)
        {
            double lat1 = latitude * Math.PI / 180;
            double lat2 = node.GetLatitude() * Math.PI / 180;

            double lonDistance = (longitude - node.GetLongitude()) * Math.PI /
180;
            double latDistance = (latitude - node.GetLatitude()) * Math.PI /
180;

            double a = (Math.Sin(latDistance / 2) * Math.Sin(latDistance / 2) +
                        Math.Cos(lat1) * Math.Cos(lat2) *
                        Math.Sin(lonDistance / 2) * Math.Sin(lonDistance / 2));

            double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
            return Math.Round(c * 6371 * 1000);
        }

        //Getter
        public string GetName()
        {
            return name;
        }

        public double GetLongitude()
        {
            return longitude;
        }

        public double GetLatitude()
        {
            return latitude;
        }

        public int GetID()
        {
            return id;
        }
```

```
        public List<int> GetAdjList()
        {
            return adjList;
        }

        //Setter
        public void SetEstimatedDistance(double est)
        {
            estimatedDistance = est;
        }

        public void SetDistanceFromStart(double dist)
        {
            distanceFromStart = dist;
        }

        public void SetParentID(int Pid)
        {
            parentID = Pid;
        }

        public double GetEstimatedDistance()
        {
            return estimatedDistance;
        }

        public double GetDistanceFromStart()
        {
            return distanceFromStart;
        }

        public int GetParentID()
        {
            return parentID;
        }

        public bool GetVisited()
        {
            return visited;
        }

        public void Visit()
        {
            visited = true;
        }

        public void UnVisit()
        {
            visited = false;
        }
    }
}
```

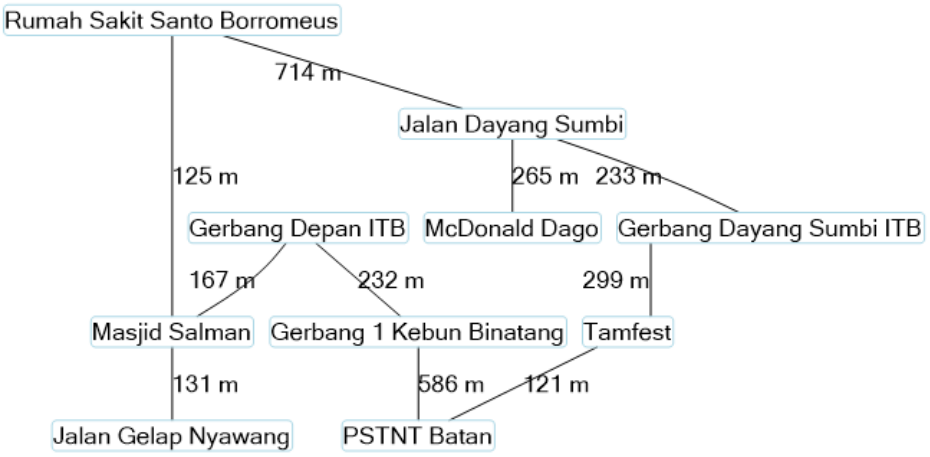
1.5. SortedQueue.cs

```
using System;
```

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

namespace PathFinder
{
    class SortedQueue
    {
        private List<Node> queue;

        public SortedQueue()
        {
            queue = new List<Node>();
        }
        public void Enqueue(Node node)
        {
            int i = 0;

            while (i < queue.Count && queue[i].GetEstimatedDistance() <
                   node.GetEstimatedDistance())
            {
                i++;
            }

            queue.Insert(i, node);
        }

        public Node Dequeue()
        {
            if (queue.Count == 0) return null;
            Node first = queue[0];
            queue.RemoveAt(0);
            return first;
        }
    }
}
```

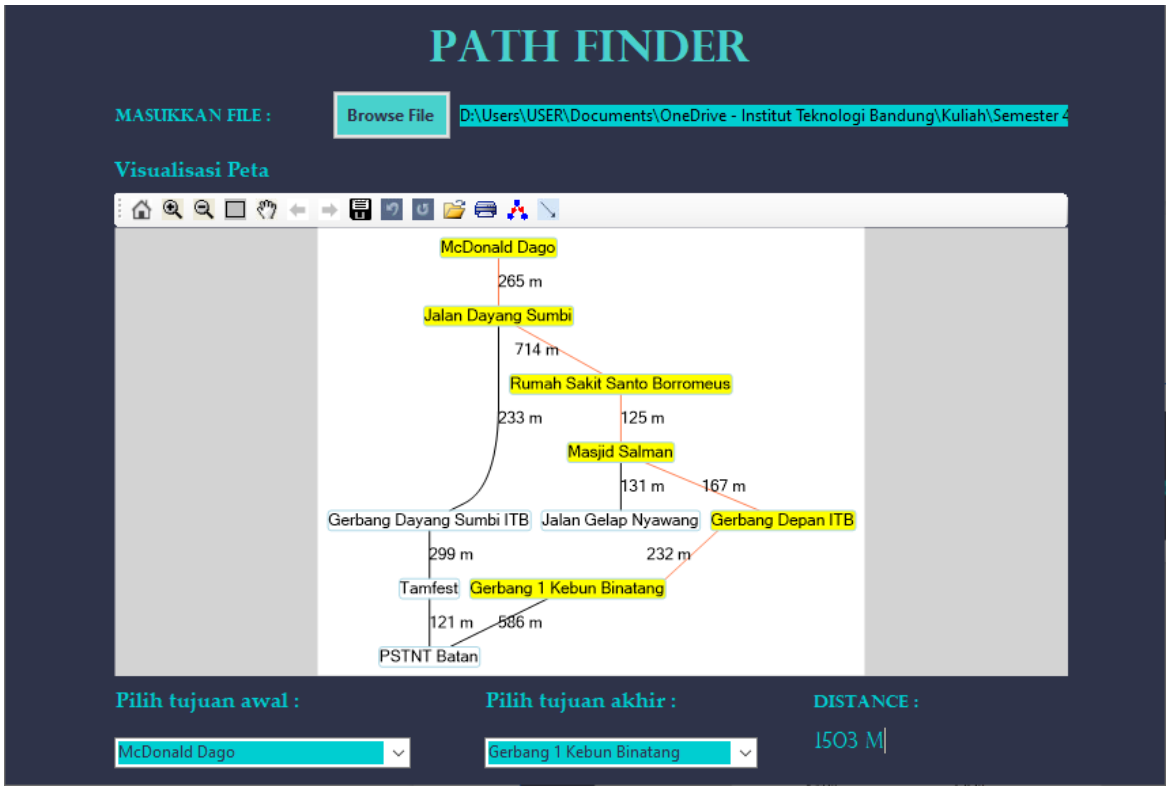
2. Hasil Eksekusi Program

2.1. Peta 1 (Jalan Sekitar ITB)



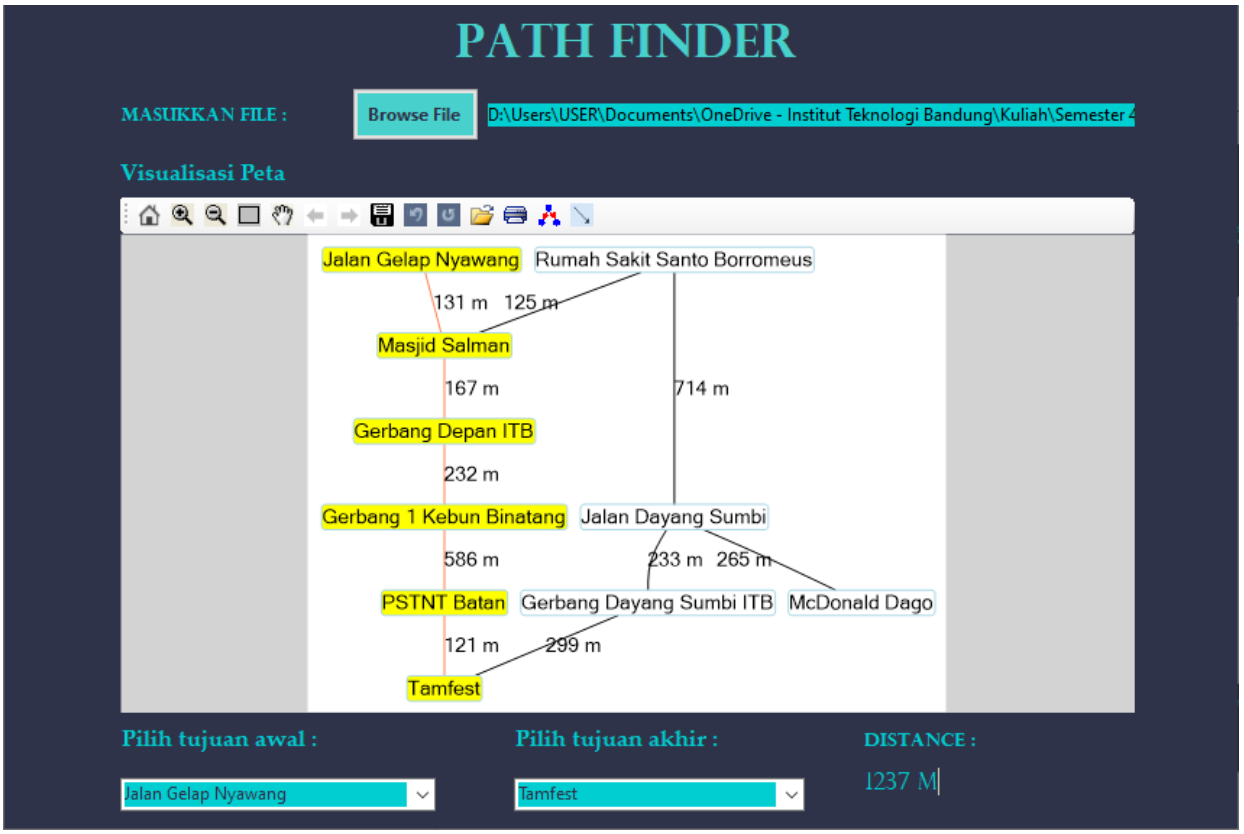
Gambar 2.1.1 Peta Jalan Sekitar ITB

2.1.1. Eksekusi 1



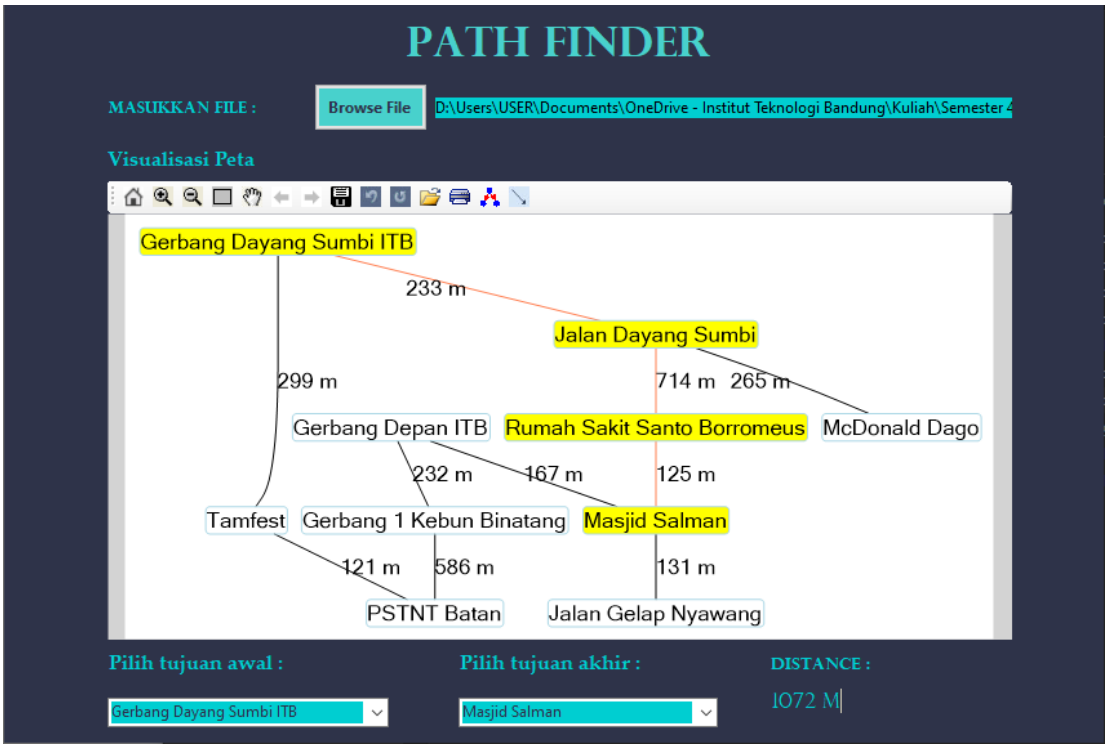
Gambar 2.1.1.1 Hasil Eksekusi 1 Peta 1

2.1.2. Eksekusi 2



Gambar 2.1.2.1 Hasil Eksekusi 2 Peta 1

2.1.3. Eksekusi 3

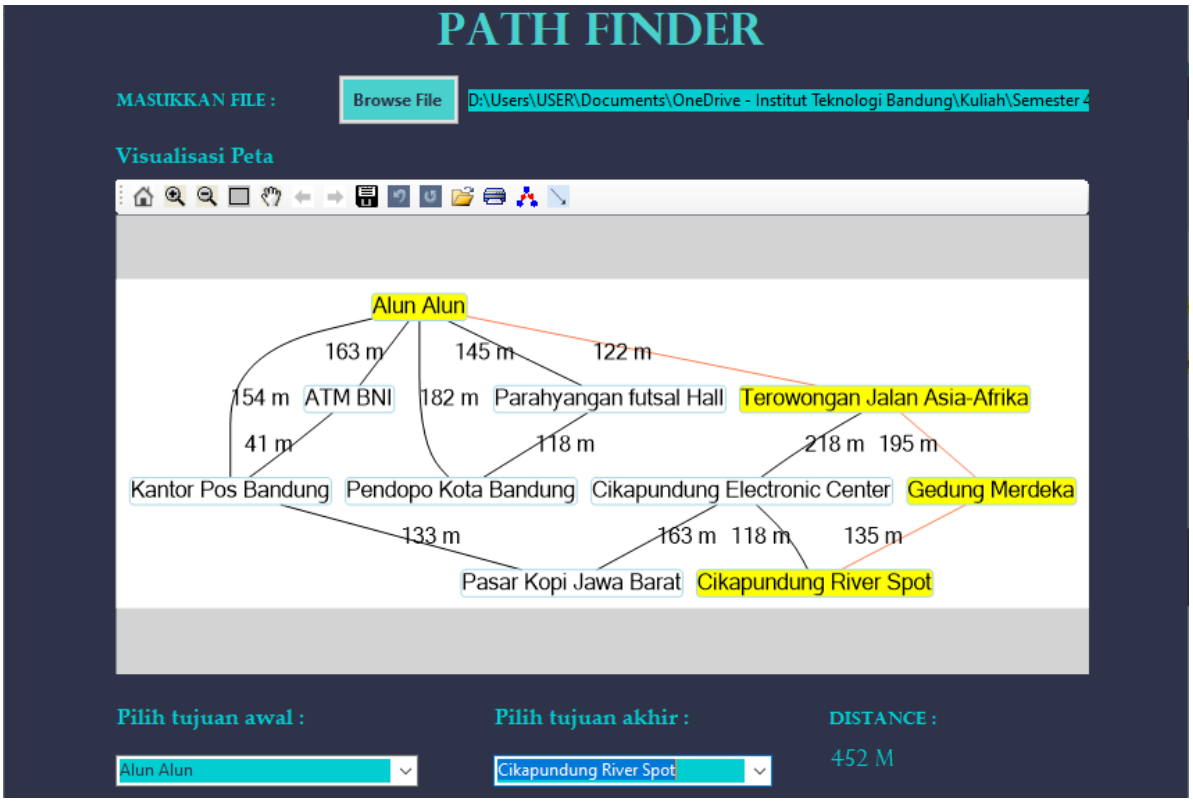


Gambar 2.1.3.1 Hasil Eksekusi 3 Peta 1

2.2. Peta 2 (Peta Alun-Alun Bandung)

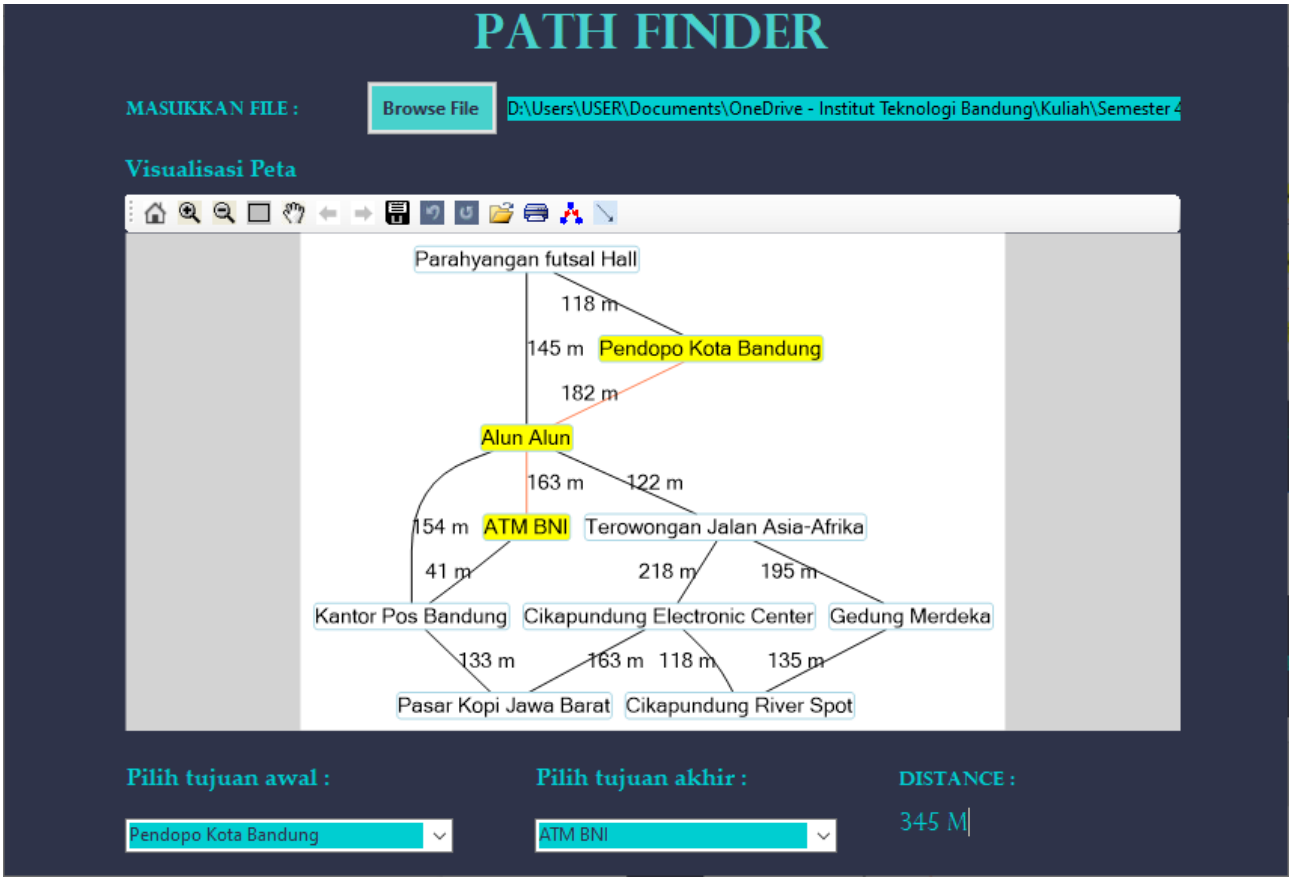
Gambar 2.2.1 Peta Alun-Alun Bandung

2.2.1. Eksekusi 1



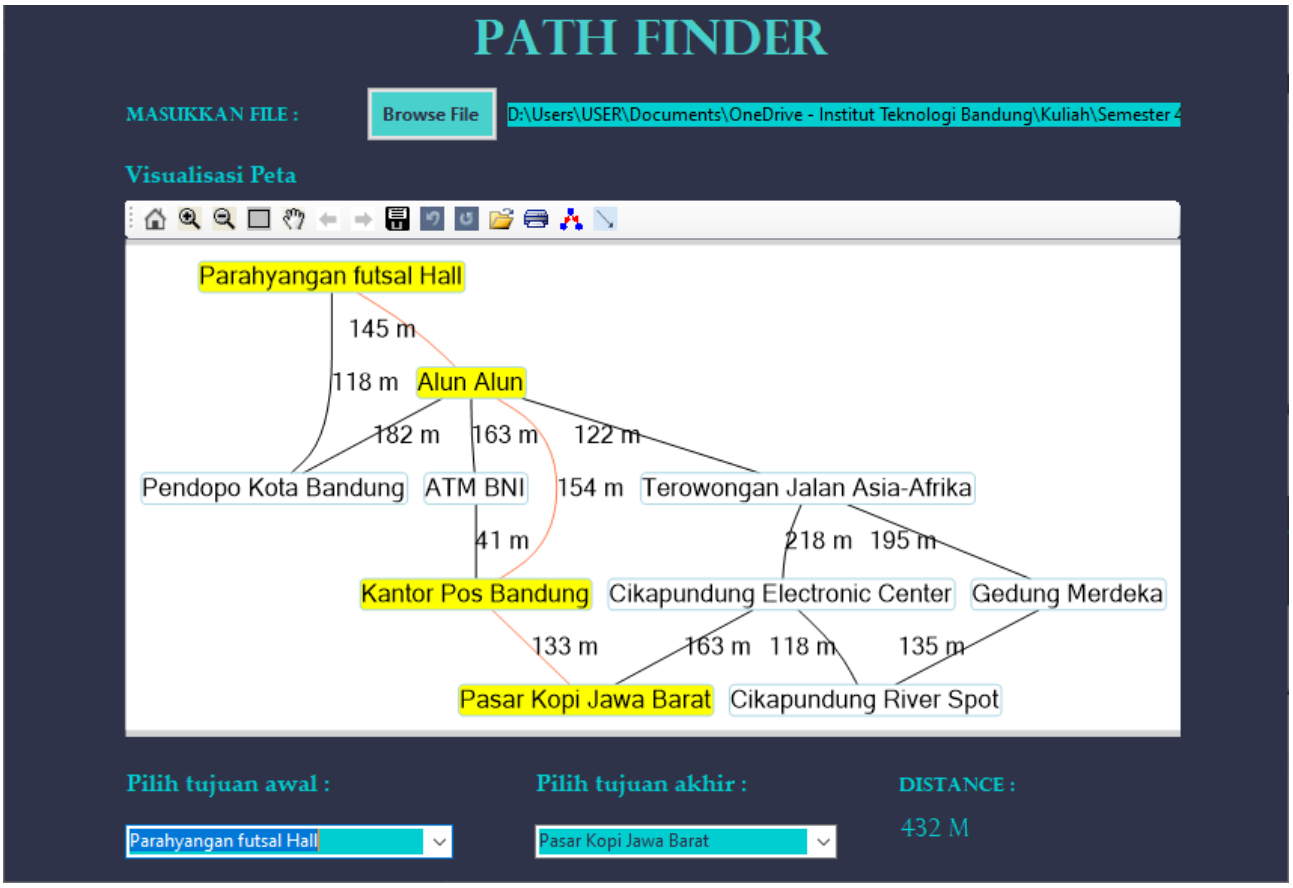
Gambar 2.2.1.1 Hasil Eksekusi 1 Peta 2

2.2.2. Eksekusi 2



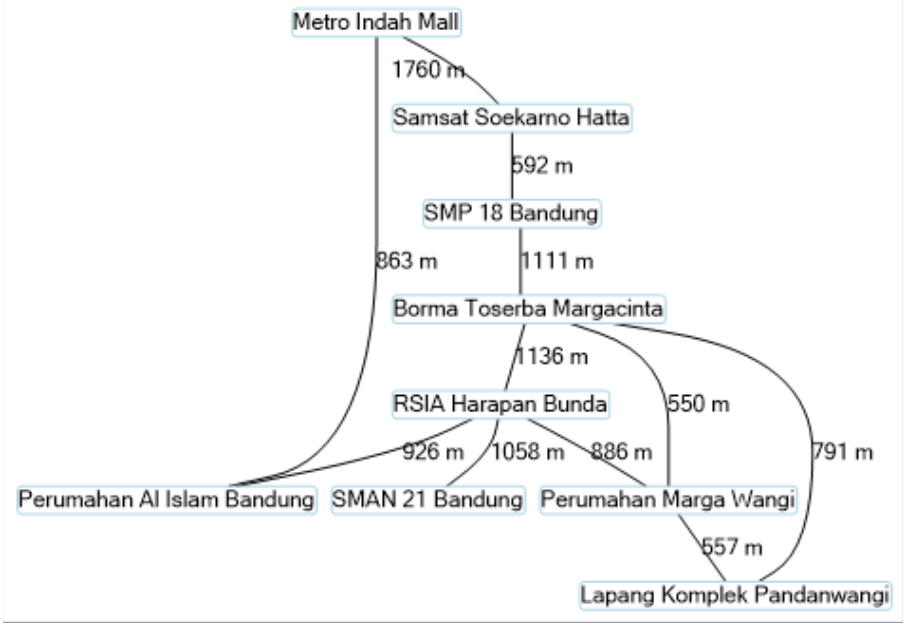
Gambar 2.2.2.1 Hasil Eksekusi 2 Peta 2

2.2.3. Eksekusi 3



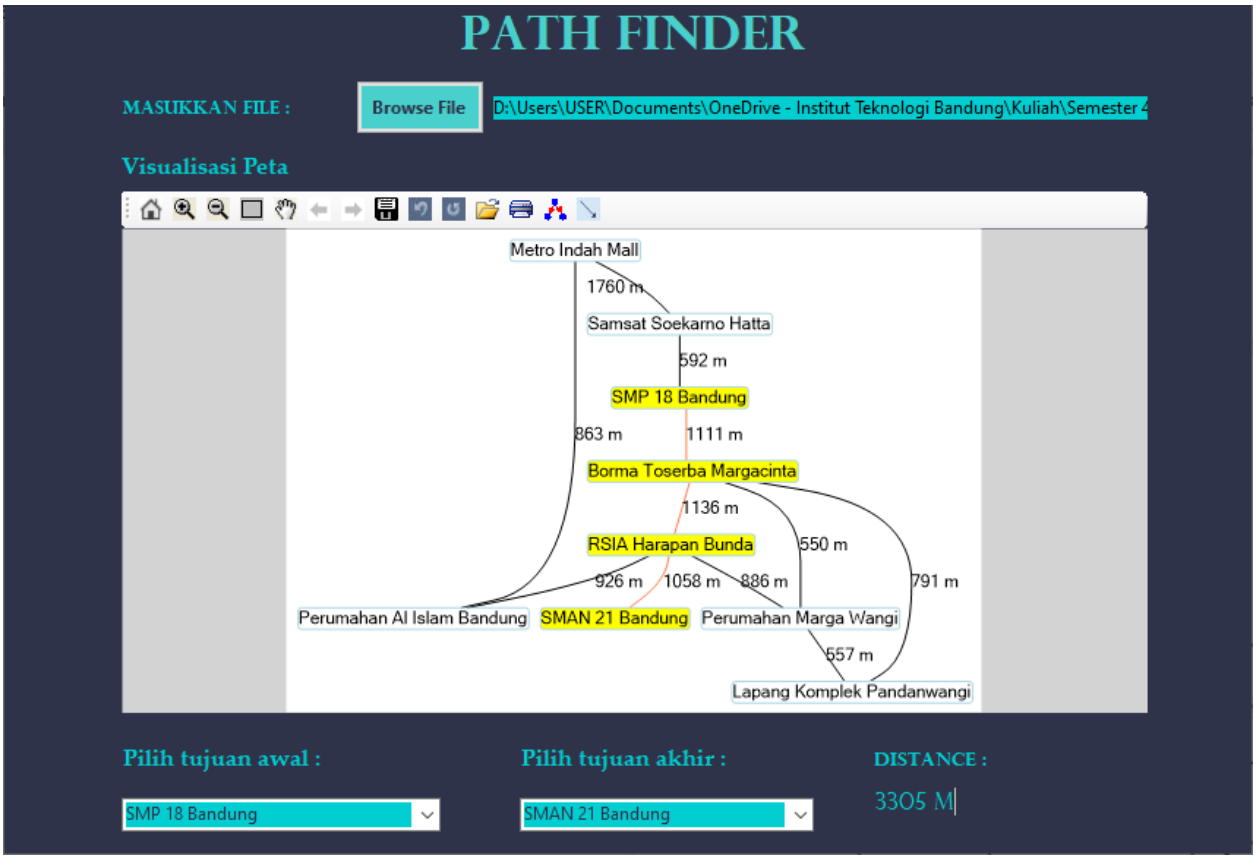
Gambar 2.2.3.1 Hasil Eksekusi 3 Peta 2

2.3. Peta 3 (Peta Buahbatu)



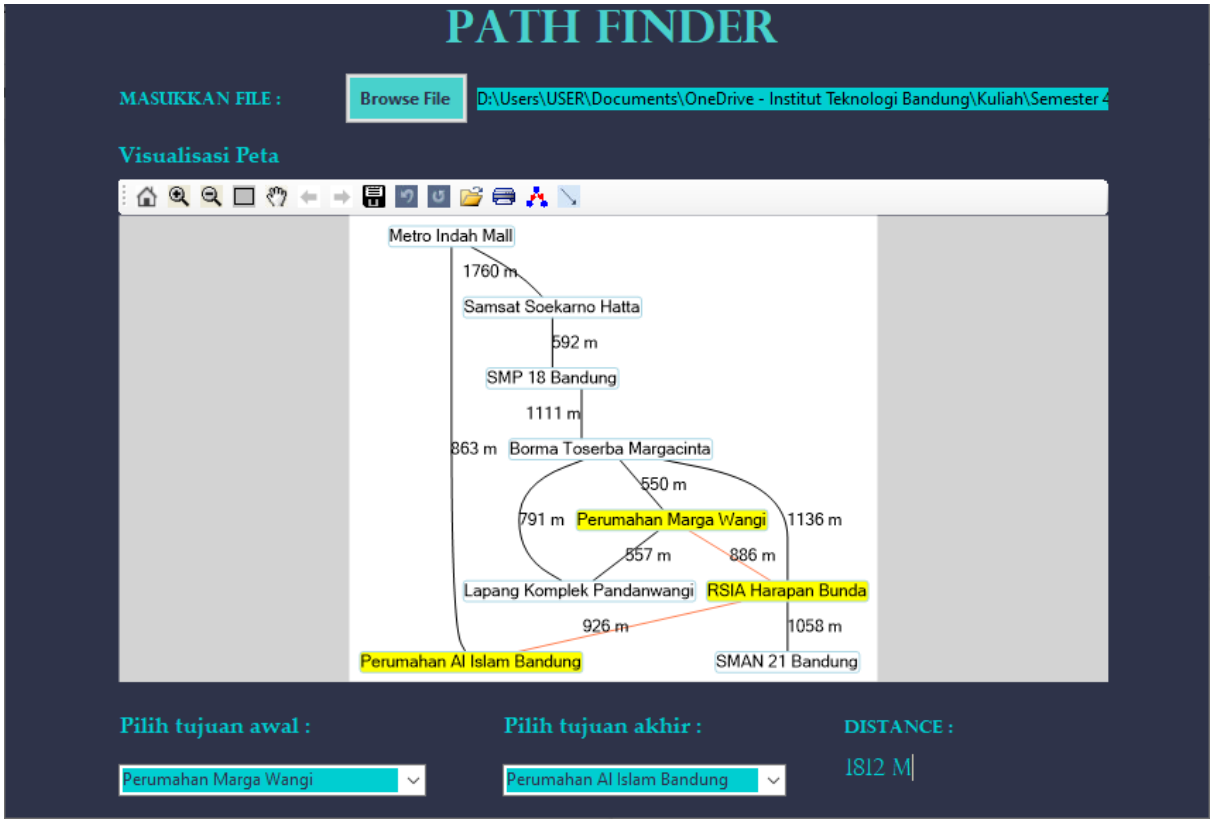
Gambar 2.3.1 Peta Buahbatu

2.3.1. Eksekusi 1



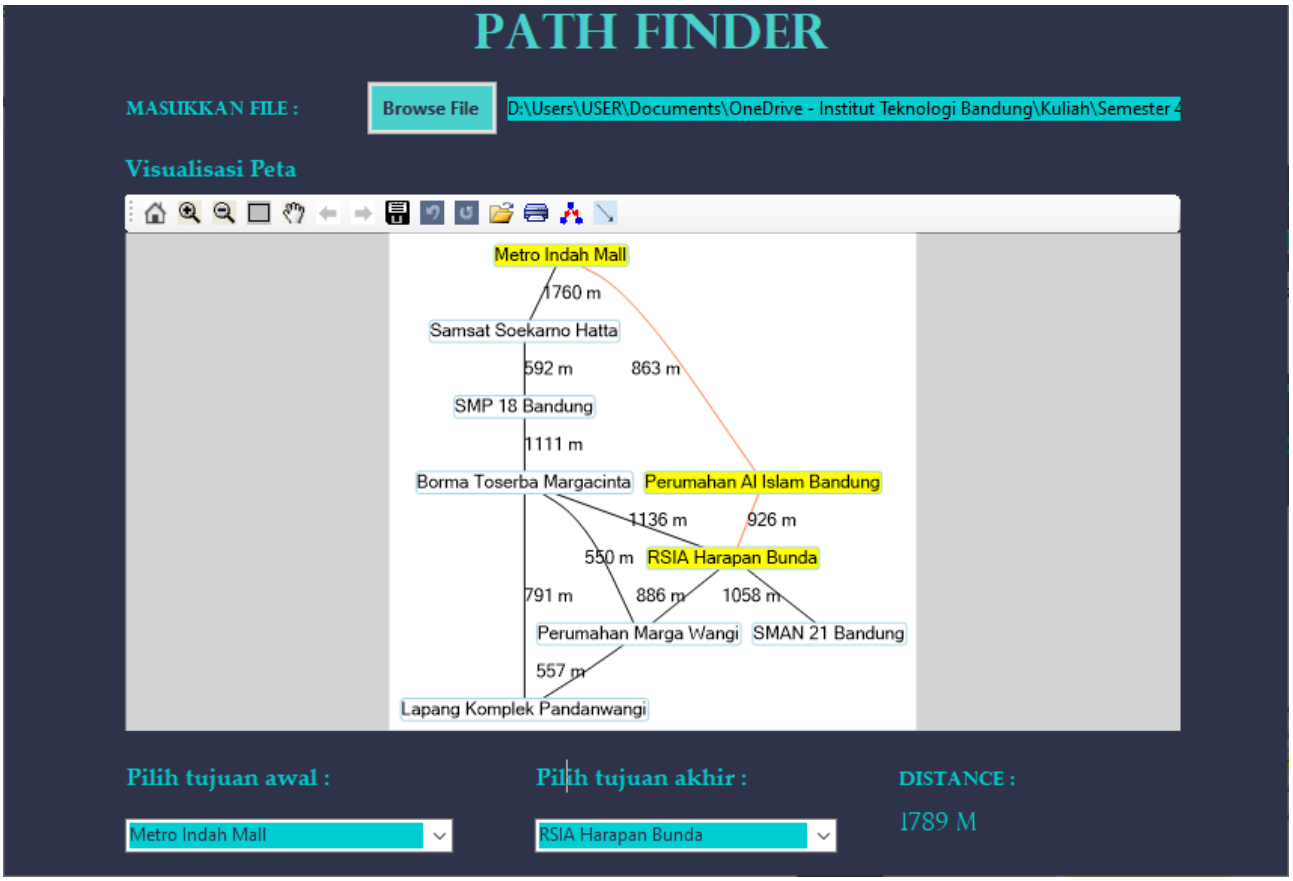
Gambar 2.3.1.1 Hasil Eksekusi 1 Peta 3

2.3.2. Eksekusi 2



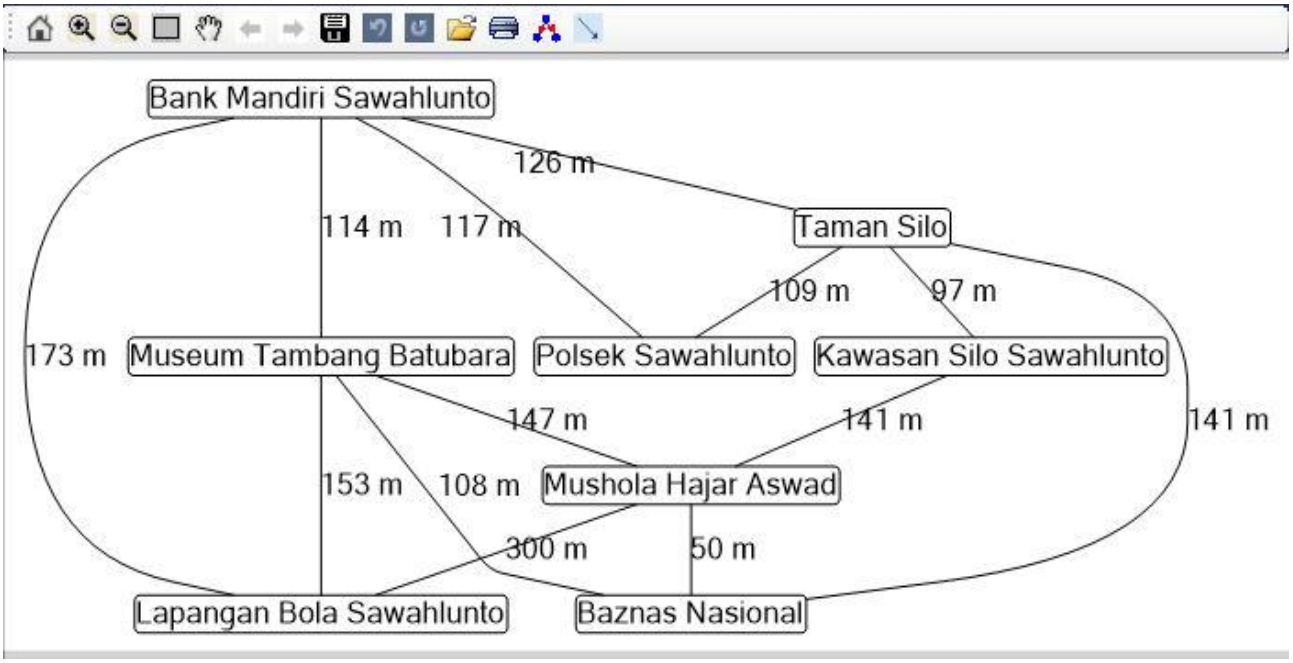
Gambar 2.3.2.1 Hasil Eksekusi 2 Peta 3

2.3.3. Eksekusi 3



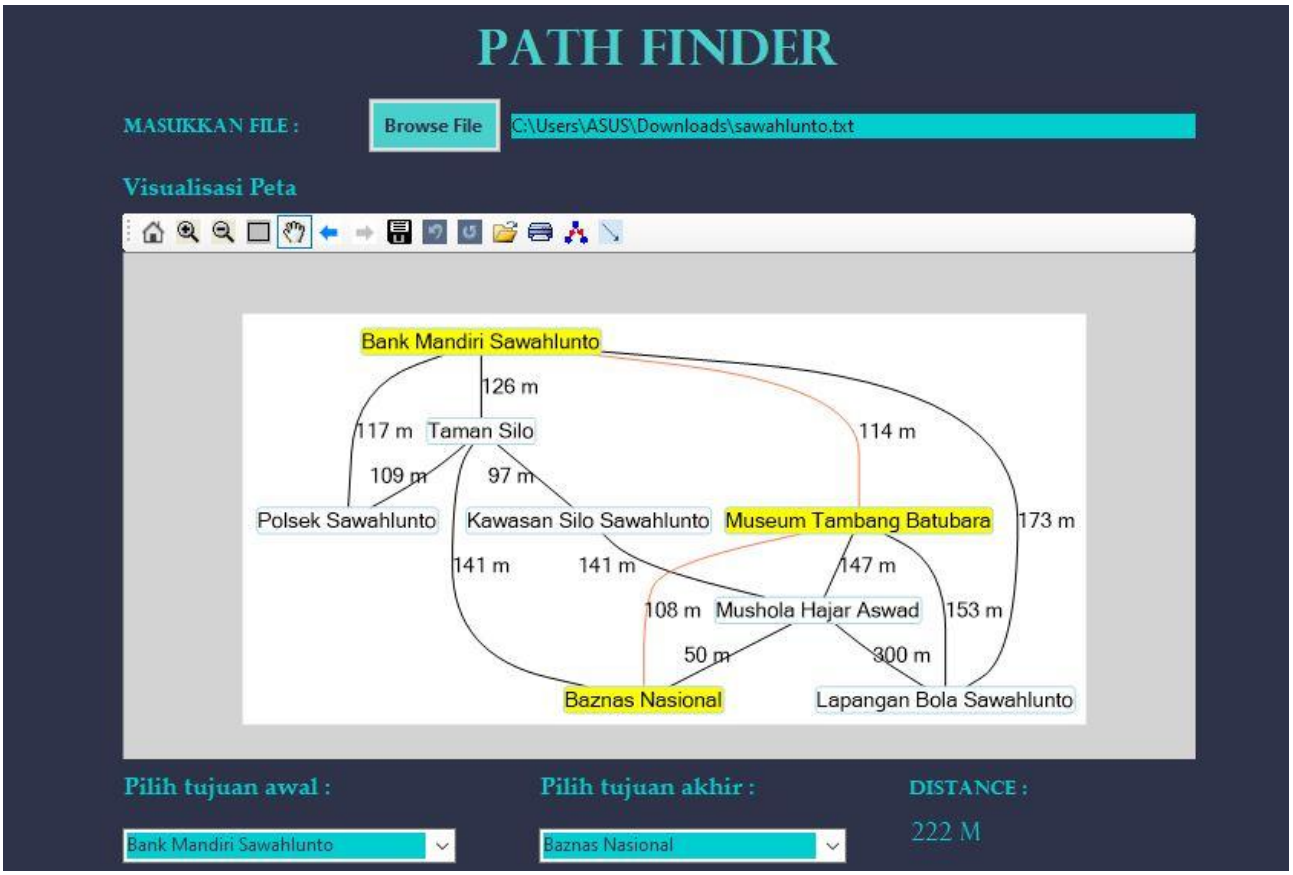
Gambar 2.3.3.1 Hasil Eksekusi 3 Peta 3

2.4. Peta 4(Peta Sawahlunto)



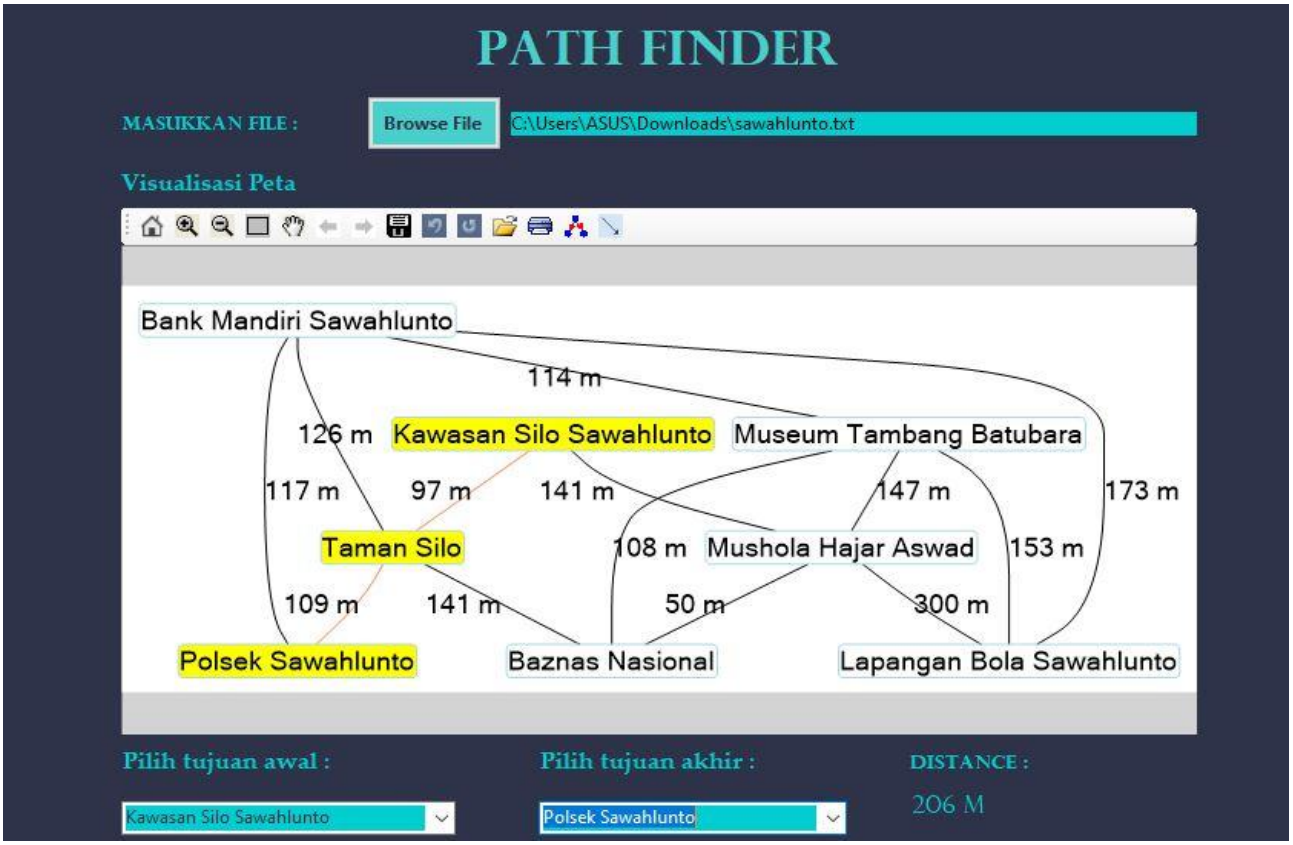
Gambar 2.4.1. Peta Sawahlunto

2.4.1. Eksekusi 1



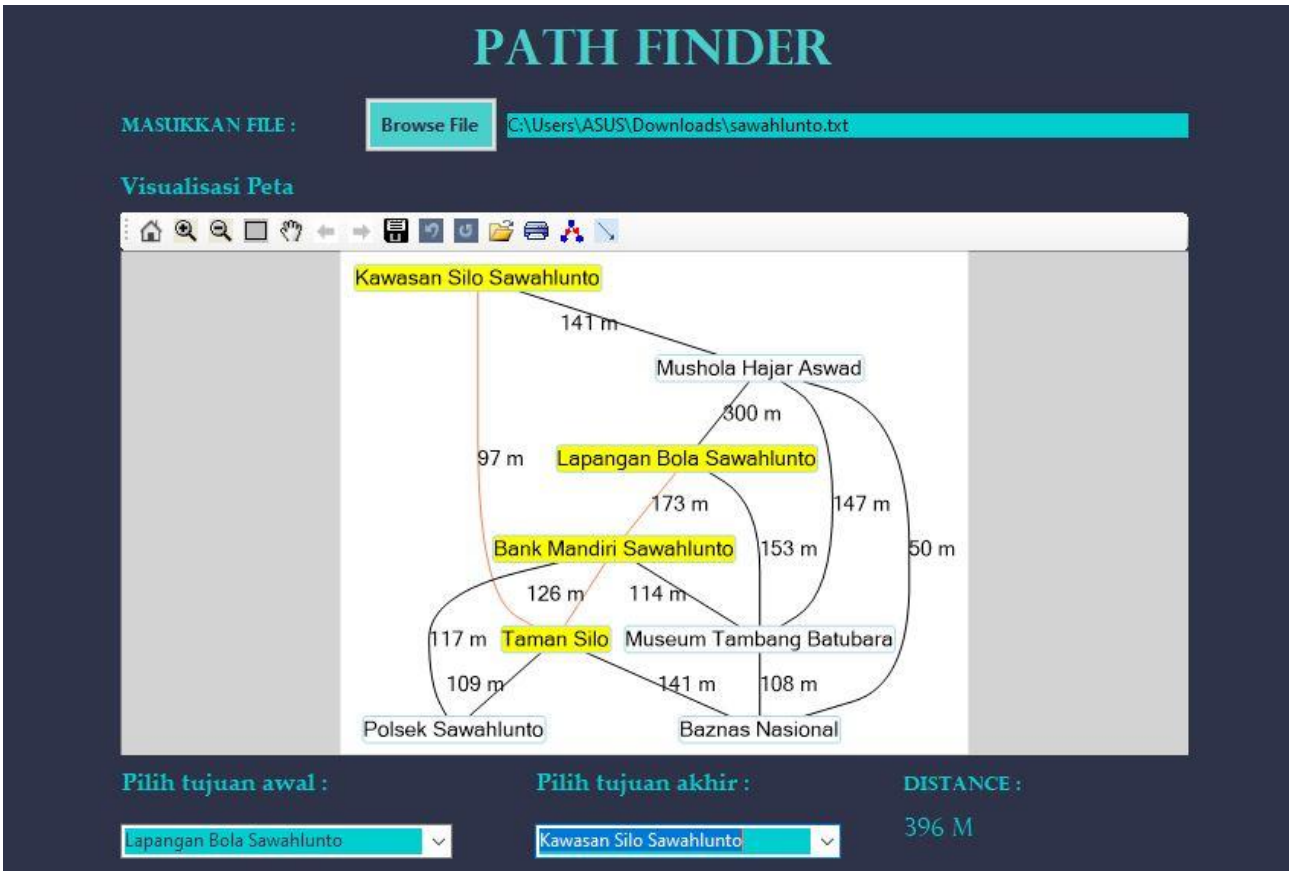
Gambar 2.4.1.1 Eksekusi 1 Peta 4

2.4.2. Eksekusi 2



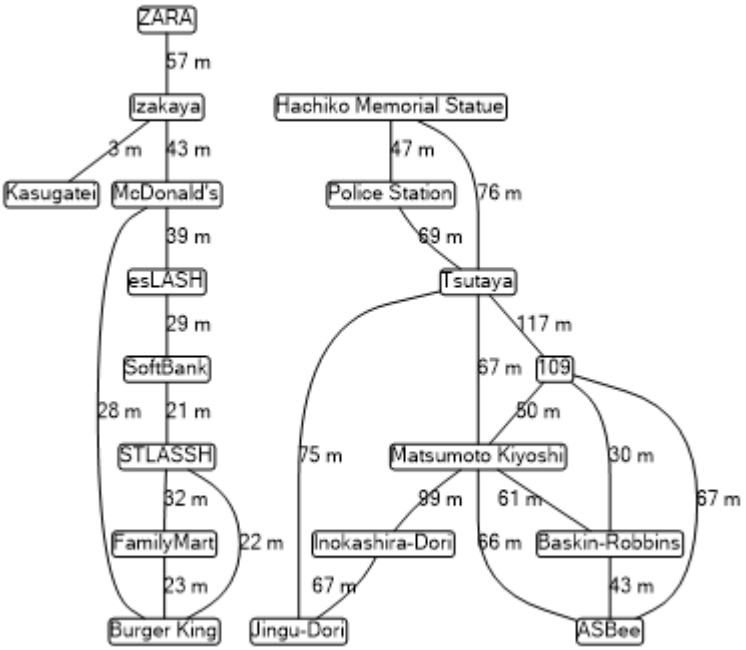
Gambar 2.4.1.2 Eksekusi 2 Peta 4

2.4.3. Eksekusi



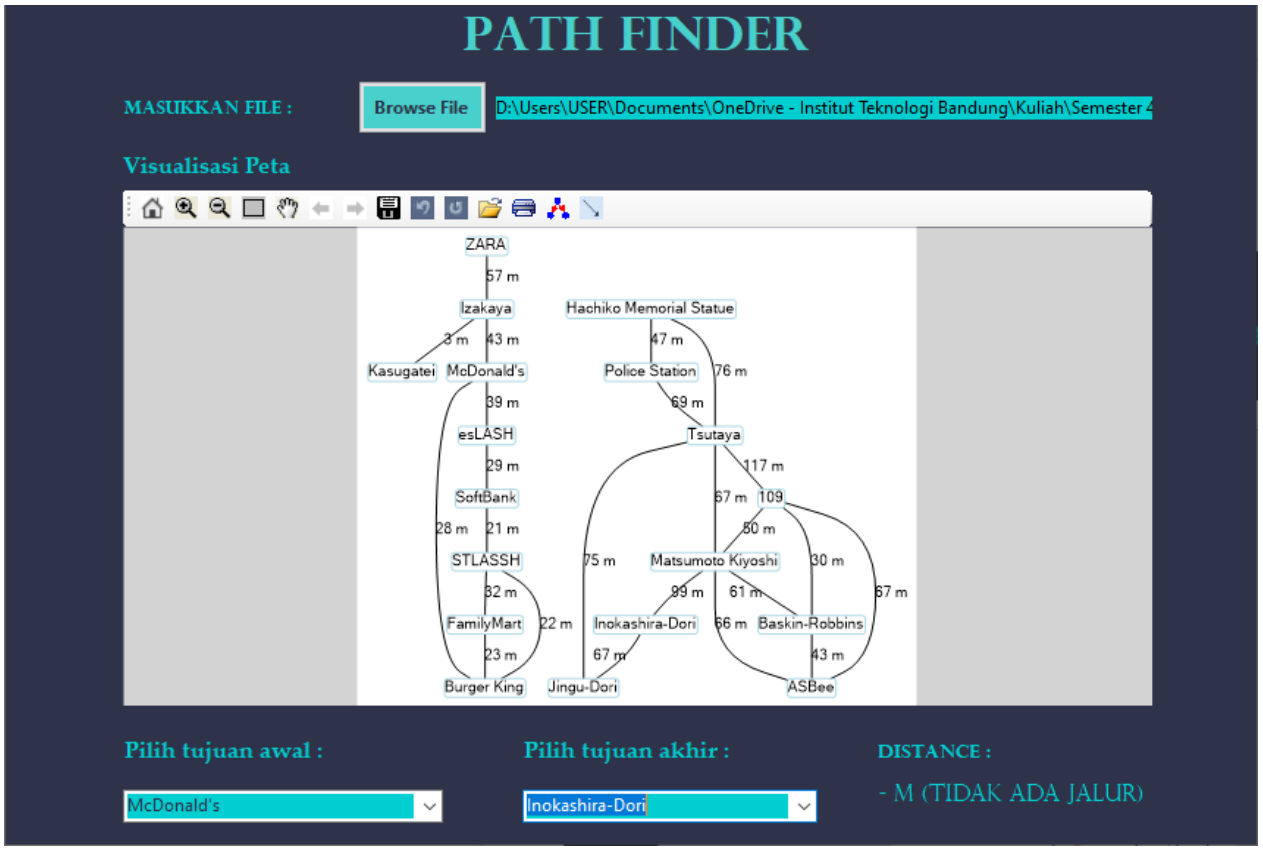
Gambar 2.4.1.3 Eksekusi 3 Peta 4

2.5. Peta 5 (Peta Shibuya)



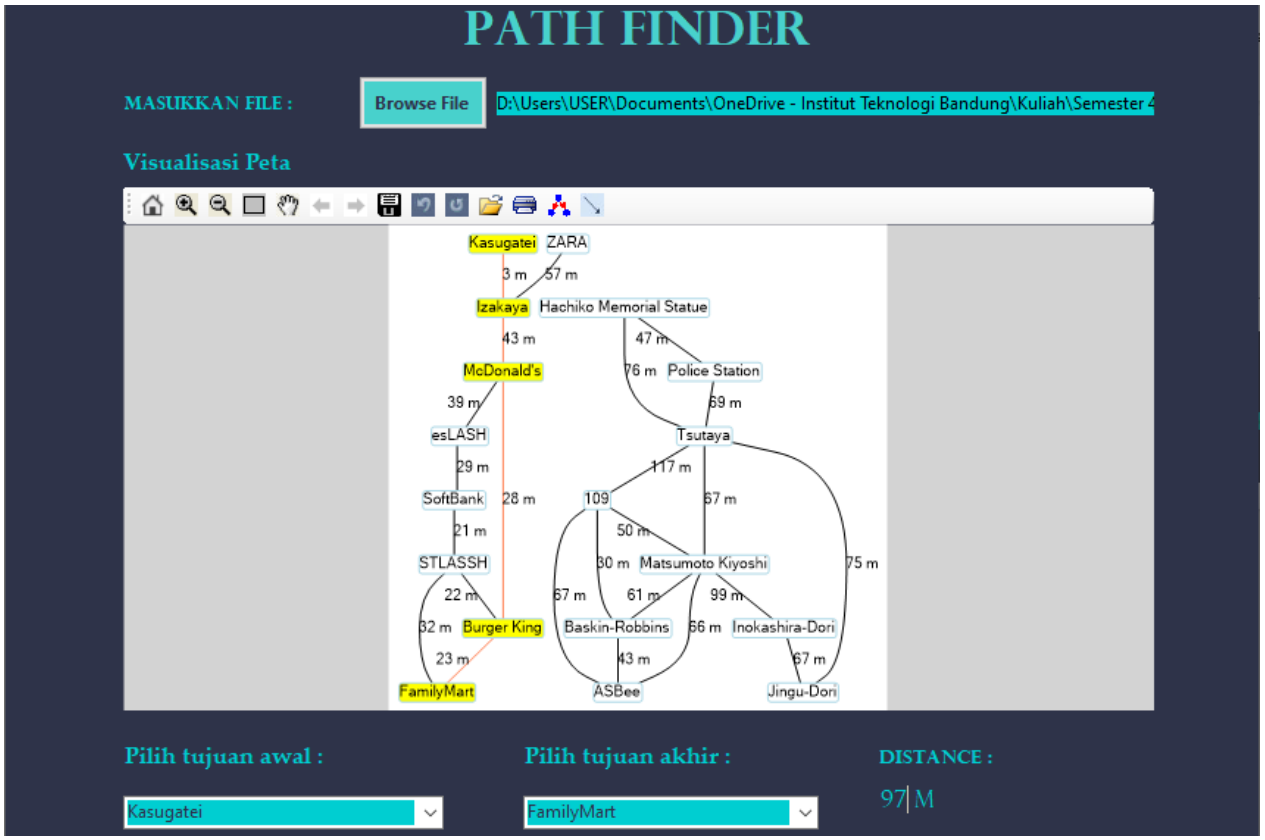
Gambar 2.5.1 Peta Shibuya

2.5.1. Eksekusi 1



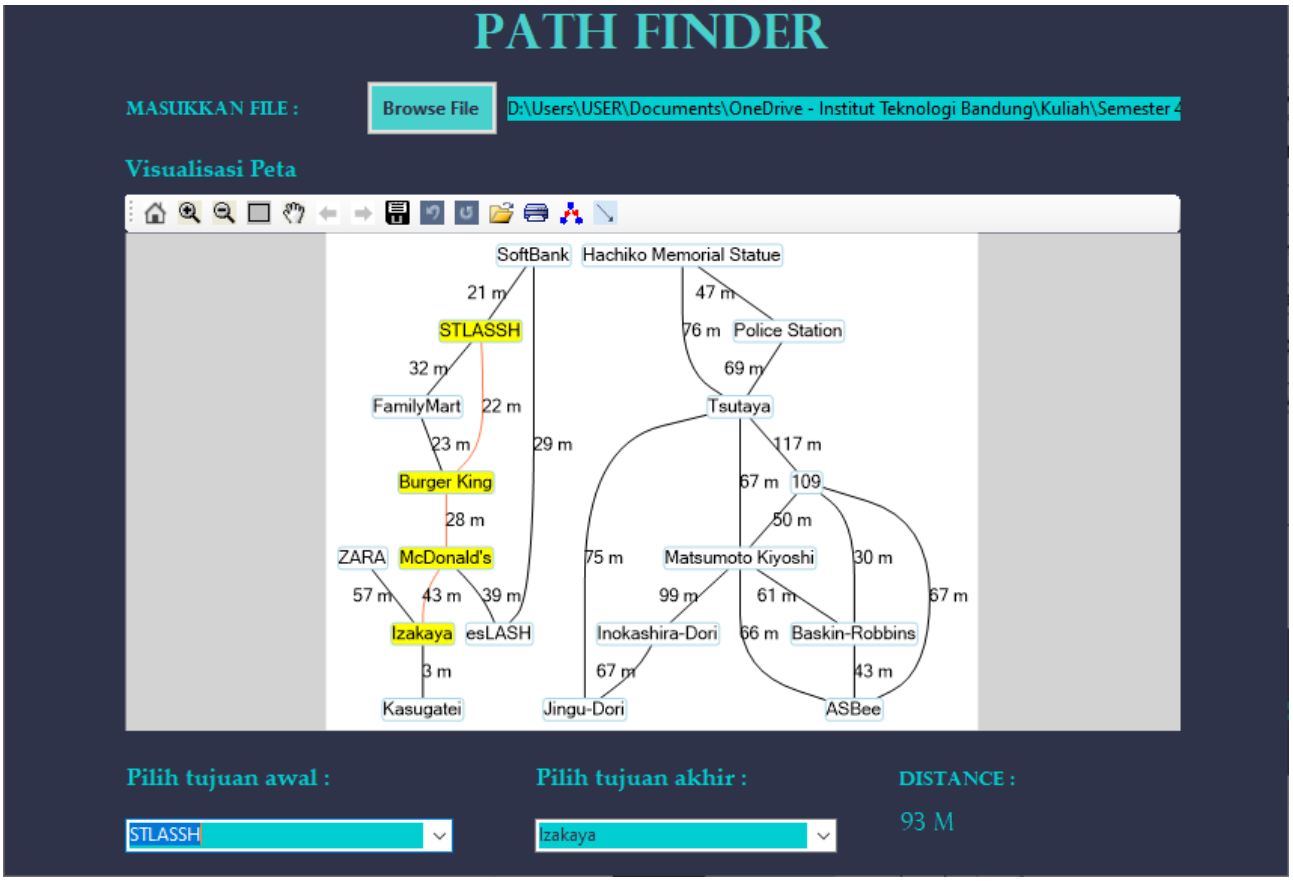
Gambar 2.5.1.1 Eksekusi 1 Peta 5

2.5.2. Eksekusi 2



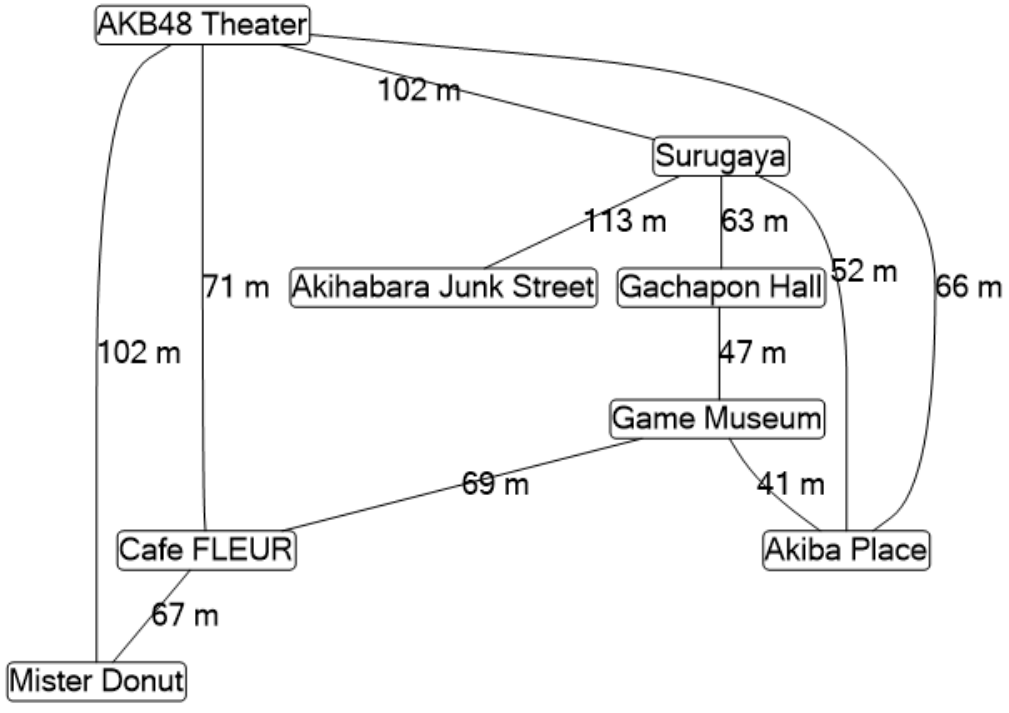
Gambar 2.5.2.1 Eksekusi 2 Peta 5

2.5.3. Eksekusi 3



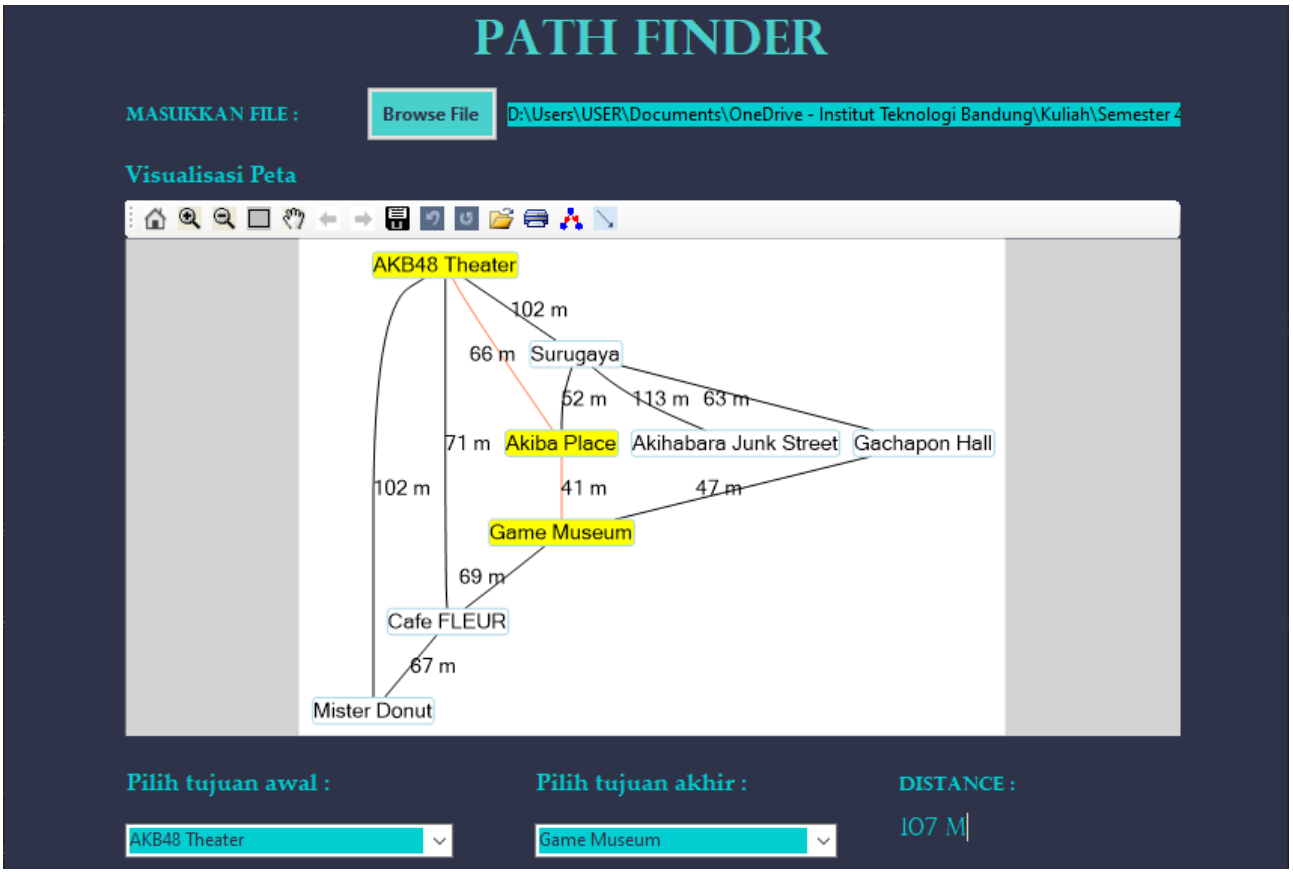
Gambar 2.5.3.1 Eksekusi 3 Peta 5

2.6. Peta 6 (Akihabara)



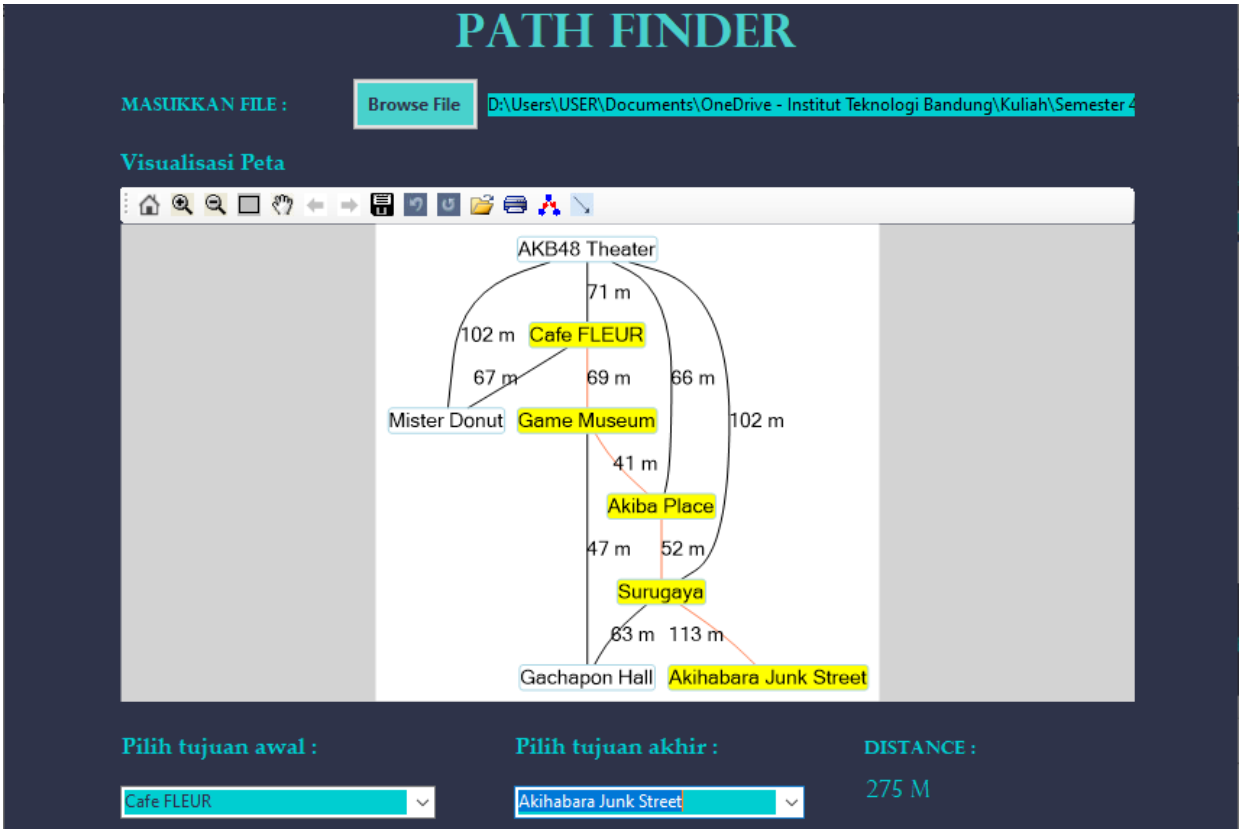
Gambar 2.6.1 Peta Akihabara

2.6.1. Eksekusi 1



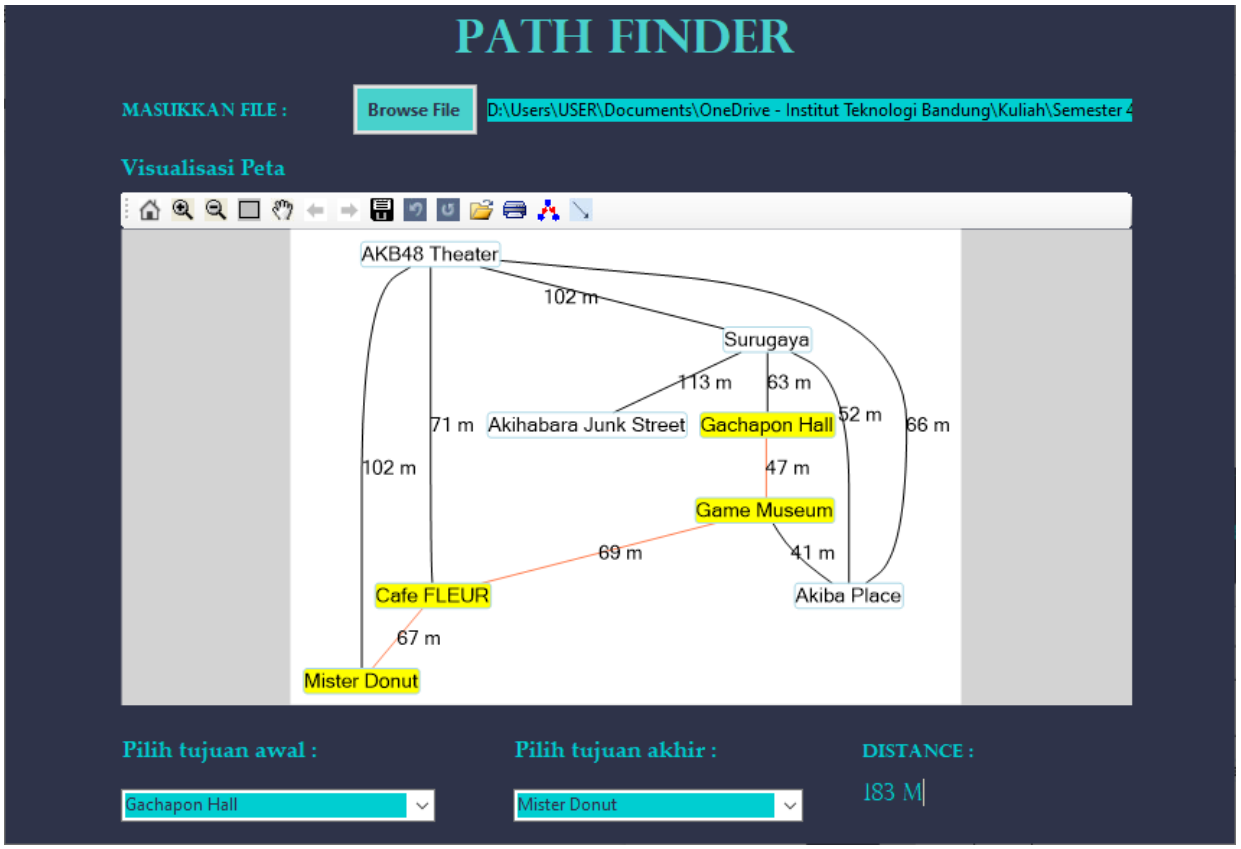
Gambar 2.6.1.1 Eksekusi 1 Peta 6

2.6.2. Eksekusi 2



Gambar 2.6.2.1 Eksekusi 2 Peta 6

2.6.3. Eksekusi 3



Gambar 2.6.3.1 Eksekusi 3 Peta 6

3. Alamat untuk Kode Program

https://github.com/wundersmith/Tucil3-Stima-PathFinder

4. Cek List

Poin	Ya	Tidak
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		✓