Tugas Kecil 3
Penentuan Rute Terpendek menggunakan Algoritma UCS dan A*
IF2211 Strategi Algoritma

Dibuat Oleh:

Matthew Mahendra	13521007
Christophorus Dharma Winata	13521009

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2023

Daftar Isi

BAB 1

Latar Belakang

1.1 Deskripsi Persoalan

Pada peta, kadang kala diperlukan rute yang terpendek agar dapat menghemat perjalanan, baik dari segi biaya maupun tenaga. Pencarian rute terpendek ini dapat menggunakan beberapa algoritma pada graf. Contoh dari algoritma tersebut adalah algoritma A* dan Uniformed Cost Search.

1.2 Algoritma Uniformed Cost Search (UCS)

Algoritma Uniformed Cost Search (UCS) merupakan modifikasi dari breadth-first search dan iterative depth search, tetapi menghasilkan langkah terpendek yang paling memungkinkan. Algoritma ini memperhitungkan biaya atau *cost* dari setiap simpul ke simpul lainnya pada saat memeriksa simpul-simpul yang bertetangga. Setelahnya, dari simpul-simpul tersebut, diambil simpul dengan nilai terkecil.

Dengan memeriksa simpul dengan nilai yang paling kecil terlebih dahulu, dapat dipastikan bahwa langkah yang diambil akan menghasilkan alur pergerakan yang paling hemat dan juga cost yang hemat. Fungsi untuk menghitung cost diberi nama $g(n)$ yang digunakan untuk mengukur cost dari suatu simpul ke simpul lainnya.

1.3 Algoritma A*

Algoritma A* merupakan bentuk informed search dari algoritma UCS yang menggunakan nilai heuristic estimasi jarak lurus dari suatu simpul ke simpul tujuan. Pemeriksaan tidak hanya menggunakan nilai $g(n)$ tetapi juga nilai dari $h(n)$ yang merupakan nilai heuristic seperti yang sudah dijelaskan.

Dengan menggunakan perhitungan ini, akan dihasilkan alur pergerakan yang lebih optimal lagi dikarenakan adanya tambahan informasi dari $h(n)$ untuk penentuan nilai terkecil pada setiap pemeriksaan simpul.

BAB 2

Hasil

2.1 Penerapan UCS

Secara umum, penerapan UCS dalam penentuan rute terpendek adalah sebagai berikut,

1. Dari simpul awal, catatlah semua simpul yang bertetangga, beserta nilai jarak dari simpul awal ke simpul yang bertetangga

2. Urutkan simpul-simpul berdasarkan nilai jarak terkecil

3. Dari simpul yang memiliki nilai terkecil, catat kembali semua simpul yang bertetangga dan jumlahkan nilai jarak dari simpul tersebut ke simpul yang bertetangga dengan nilai sebelumnya

4. Proses dilangsungkan kembali hingga tercapai simpul tujuan

2.2 Penerapan A*

Secara umum, penerapan A* dalam penentuan rute terpendek adalah sebagai berikut,

1. Dari simpul awal, catatlah semua simpul yang bertetangga, beserta nilai jarak dari simpul awal ke simpul yang bertetangga ($g(n)$) yang dijumlahkan dengan jarak lurus dari simpul awal ke simpul tujuan ($h(n)$)

2. Urutkan simpul-simpul berdasarkan nilai $g(n) + h(n)$

3. Dari simpul yang memiliki nilai terkecil, catat kembali semua simpul yang bertetangga dan jumlahkan nilai $g(n)$ jarak dari simpul tersebut ke simpul yang bertetangga dengan nilai sebelumnya serta catat jarak dari simpul tersebut ke simpul tujuan

4. Proses dilangsungkan kembali hingga tercapai simpul tujuan

2.3 Source Code

Program dibagi menjadi beberapa file yaitu, Solver.java, Graph.java, Location.java, Node.java, dan App.java. App.java adalah file yang digunakan untuk menjalankan program

2.3.1 App.java

```
package stima;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import javax.swing.JFrame;

import org.jxmapviewer.JXMapViewer;
import org.jxmapviewer.OSMTileFactoryInfo;
import org.jxmapviewer.painter.CompoundPainter;
import org.jxmapviewer.painter.Painter;
import org.jxmapviewer.viewer.DefaultTileFactory;
import org.jxmapviewer.viewer.DefaultWaypoint;
import org.jxmapviewer.viewer.GeoPosition;
import org.jxmapviewer.viewer.TileFactoryInfo;
import org.jxmapviewer.viewer.Waypoint;
import org.jxmapviewer.viewer.WaypointPainter;

import algorithms.*;
import visuals.RoutePainter;

/**
 * Aplikasi yang menerima file input graf map
 * dan menampilkan hasil path terpendek dari point start ke point finish
 * @author Matthew Mahendra
 * @author Christophorus Dharma Winata
 */
public class App
{
    /**
     * @param args the program args (ignored)
     */
    public static void main(String[] args)
    {
        // Opening java terminal
        System.out.println(”Welcome to the shortest path finder!”);
        System.out.println(”Please enter the file name of the map you want to use:”);
        // Input file name
        String fileName = System.console().readLine();

        // Read file and create graph
        Graph graph = new Graph(fileName);
        // Print location
        System.out.println("\nMAP LOCATIONS: ");
        for (int i = 0; i < graph.getLocCount(); i++) {
            System.out.println((i+1) + ". " + graph.getLocName(i));
        }

        //input start and finish location
        System.out.println("\nEnter the starting location name:");
        String startingPosition = System.console().readLine();
        System.out.println("\nEnter the target finish location name:");
        String finishPosition = System.console().readLine();

        // Calling solver from algorithms
        Solver _solver = new Solver(startingPosition, finishPosition, fileName);

        // Choosing algorithm
        System.out.println("\nPlease choose the algorithm for pathfinding:");
        System.out.println("1. UCS");
        System.out.println("2. A*");
        int choice = Integer.parseInt(System.console().readLine());
        ArrayList<String> path;
        if (choice == 1) {
            // Path and distance from UCS algorithm
            path = _solver.UCS();
        } else if (choice == 2) {
            // Path and distance from A* algorithm
            path = _solver.AStar();
        } else {
            System.out.println("Invalid choice");
            return;
        }

        System.out.println("Shortest Path:");
        for(int i = 0; i < path.size(); i++){
            if(i != path.size() - 1){
                System.out.print(path.get(i) + " - ");
            }else{
                System.out.println(path.get(i));
            }
        }
        System.out.println("Distance: " + _solver.getJarak());

        // Instantiate JXMapViewer
        JXMapViewer mapViewer = new JXMapViewer();

        // Display the viewer in a JFrame
        JFrame frame = new JFrame(”Map Viewer”);
        frame.getContentPane().add(mapViewer);
        frame.setSize(800, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        // Create a TileFactoryInfo for OpenStreetMap
        TileFactoryInfo info = new OSMTileFactoryInfo();
        DefaultTileFactory tileFactory = new DefaultTileFactory(info);
        mapViewer.setTileFactory(tileFactory);

        // Instantiating locations from input file
        GeoPosition[] locationsOnMap = new GeoPosition[graph.getLocCount()];
        for (int i = 0; i < graph.getLocCount(); i++) {
            locationsOnMap[i]= new GeoPosition(graph.getPos(i));
        }

        // Create a track from the geo-positions
        List<GeoPosition> solvedPath = new ArrayList<GeoPosition>();

        for (int i = 0; i < path.size(); i++) {
            solvedPath.add(new GeoPosition(graph.getPos(path.get(i))));
        }
        // Calling RoutePainter from visuals
        RoutePainter routePainter = new RoutePainter(solvedPath);

        // Set the focus
        double frac = 0.1;
        if(frac * path.size() <= 1){
            mapViewer.zoomToBestFit(new HashSet<GeoPosition>(solvedPath), frac*path.size());
        }else{
            mapViewer.zoomToBestFit(new HashSet<GeoPosition>(solvedPath), 0.2);
        }

        // Create waypoints from the geo-positions
        List<Waypoint> waypointsList = new ArrayList<Waypoint>();
        for (int i = 0; i < graph.getLocCount(); i++) {
            waypointsList.add(new DefaultWaypoint(locationsOnMap[i]));
        }
        Set<Waypoint> waypointsSet = new HashSet<Waypoint>(waypointsList);

        // Create a waypoint painter that takes all the waypoints
        WaypointPainter<Waypoint> waypointPainter = new WaypointPainter<Waypoint>();
        waypointPainter.setWaypoints(waypointsSet);

        // Create a compound painter that uses both the route-painter and the waypoint-painter
        List<Painter<JXMapViewer>> painters = new ArrayList<Painter<JXMapViewer>>();
        painters.add(routePainter);
        painters.add(waypointPainter);

        CompoundPainter<JXMapViewer> painter = new CompoundPainter<JXMapViewer>(painters);
        mapViewer.setOverlayPainter(painter);
    }
}
```

2.3.2 Solver.java

```
package algorithms;
import java.util.*;

public class Solver extends Graph{
    private PriorityQueue<Node> queue;
    private String startPoint, endPoint;
    private double jarak;

    public Solver(String sp, String ep, String fileName){
        super(fileName);
        startPoint = sp;
        endPoint = ep;
        queue = new PriorityQueue<>();
        jarak = 0;
    }

    public ArrayList<String> AStar(){
        queue = new PriorityQueue<>();

        ArrayList<String> visit = new ArrayList<>();
        visit.add(startPoint);
        Node start = new Node(startPoint, endPoint, 0, euclideanDistance(getPos(startPoint), getPos(endPoint)),visit);
        queue = new PriorityQueue<Node>();
        queue.add(start);


        while(queue.size() != 0){
            Node check = queue.remove();

            if(check.getCurrent().equals(check.getGoal())){
                this.jarak = check.calculateFN();
                return (check.getPath());
            }

            for(int i = 0; i < getNodes() ; i++){
                if(getGraph(getIndex(check.getCurrent()), i) > 0 && !check.getPath().contains(getLocName(i))){
                    /* Masukkan ke prioqueue
                    * Buat nodes baru
                    */
                    ArrayList<String> visitNew = new ArrayList<>(check.getPath());

                    visitNew.add(getLocName(i));
                    Node newNode = new Node(getLocName(i),
                                            check.getGoal(),
                                            getGraph(getIndex(check.getCurrent()), i) + check.getGn(),
                                            euclideanDistance(getPos(getLocName(i)), getPos(check.getGoal())),
                                            visitNew);
                    queue.add(newNode);
                }
            }
        }
        return new ArrayList<>();
    }

    public ArrayList<String> UCS(){
        queue = new PriorityQueue<>();

        ArrayList<String> visitedLocs = new ArrayList<>();
        visitedLocs.add(startPoint);
        Node startNode = new Node(startPoint, endPoint, 0,
           visitedLocs);
        queue.add(startNode);

        while(queue.size() != 0){
            Node check = queue.remove();

            if(check.getCurrent().equals(check.getGoal())){
                this.jarak = check.getGn();
                return (check.getPath());
            }

            for(int i = 0; i < getNodes() ; i++){
                if(getGraph(getIndex(check.getCurrent()), i) > 0
                   && !check.getPath().contains(getLocName(i)))
                  {
                    ArrayList<String> visitNew = new ArrayList
                       <>(check.getPath());

                    visitNew.add(getLocName(i));

                    Node newNode = new Node(getLocName(i),
                                            check.getGoal(),
                                            getGraph(getIndex(
                                               check.getCurrent
                                               ()), i) + check.
                                               getGn(),
                                            visitNew);

                    queue.add(newNode);
                }
            }
        }
        return new ArrayList<>();
    }

    public double getJarak(){
        return jarak;
    }
}
```

2.3.3 Node.java

```
package algorithms;
import java.util.*;

public class Node implements Comparable<Node>{
    private String current;
    private String goal;
    private double gn;
    private double hn;
    private ArrayList<String> visited = new ArrayList<>();

    /* Node untuk A* */
    public Node(String c, String g, double gn, double hn, ArrayList<String> visited){
        current=c;
        goal = g;
        this.gn = gn;
        this.hn = hn;
        this.visited = visited;
    }

    /* Node untuk UCS, tidak ada nilai h(n) */
    public Node(String c, String g, double gn, ArrayList<String> visited){
        current=c;
        goal = g;
        this.gn = gn;
        this.hn = 0;
        this.visited = visited;
    }

    public String getCurrent(){
        return current;
    }

    public String getGoal(){
        return goal;
    }

    public double getGn(){
        return gn;
    }

    public double getHn(){
        return hn;
    }

```

```java
    public double calculateFN(){
        return hn+gn;
    }

    public ArrayList<String> getPath(){
        return visited;
    }

    @Override
    public int compareTo(Node o) {
        if(calculateFN() < o.calculateFN()){
            return -1;
        }else if (calculateFN() == o.calculateFN()){
            return 0;
        }else{
            return 1;
        }
    }
}
```

2.3.4 Graph.java

```java
package algorithms;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Graph {
    protected int[][] graph;
    protected int nodes;
    protected Location[] loc;

    public Graph(String filename){
        try{
            File file = new File(”./test/”, filename);
            Scanner reader = new Scanner(file);

            /* Ambil jumlah nodes dan set ukuran matrix */
            String nString = reader.nextLine();
            int n = Integer.parseInt(nString);
            nodes = n;
            graph = new int[n][n];

            /* Insert lokasi */
            loc = new Location[n];
            for(int i = 0; i < n ; i++){
                String line = reader.nextLine();
                String[] parse = line.split(”\\s+”);
                loc[i] = new Location(parse[0], Double.parseDouble(parse[1]), Double.parseDouble(parse[2]));
            }

            /* Fill the matrix */
            for(int i = 0; i < n ; i++){
                String line = reader.nextLine();
                String[] splited = line.split(”\\s+”);
                for(int j = 0; j < n; j++){
                    graph[i][j] = Integer.parseInt(splited[j]);
                }
            }

            reader.close();

        }catch (FileNotFoundException e){
            System.out.println(”File not found!”);
            e.printStackTrace();
        }
    }
    public double euclideanDistance(double[] l1, double[] l2){
        return (Math.sqrt( Math.pow(l1[0]-l2[0], 2) + Math.pow(l1[1]-l2[1], 2) ));
    }

    public double[] getPos(String locName){
        int idx = 0;
        for(int i = 0; i < nodes; i++){
            if(loc[i].getLocName().equals(locName))
            {
                idx = i;
                break;
            }
        }
        return loc[idx].getCoord();
    }

    /**
     * Mengembalikan koordinat lokasi berdasarkan indeks
     * @param i indeks lokasi
     * @return double[] koordinat lokasi
     */
    public double[] getPos(int i){
        return loc[i].getCoord();
    }

    public String getLocName(int i){
        return loc[i].getLocName();
    }

    public int getIndex(String locName){
        int idx = 0;
        for(int i = 0; i < nodes; i++){
            if(loc[i].getLocName().equals(locName))
            {
                idx = i;
                break;
            }
        }
        return idx;
    }

    public int getGraph(int b, int c){
        return graph[b][c];
    }

    public int getNodes(){
        return nodes;
    }
    public int getLocCount(){
        return loc.length;
    }
}
```

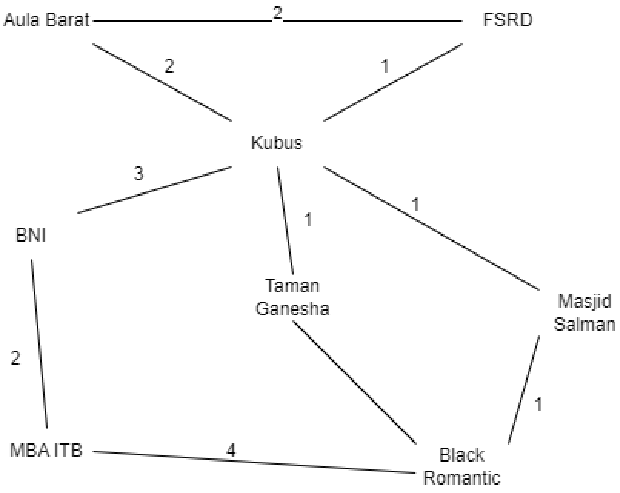
2.3.5 Location.java

```
package algorithms;
public class Location {
    private String locName;
    private double x;
    private double y;

    public Location(String locName, double x, double y){
        this.x = x;
        this.y = y;
        this.locName = locName;
    }

    public String getLocName(){
        return locName;
    }

    public double[] getCoord(){
        double[] coord = {this.x, this.y};
        return coord;
    }
}
```

2.4 Hasil Pengujian

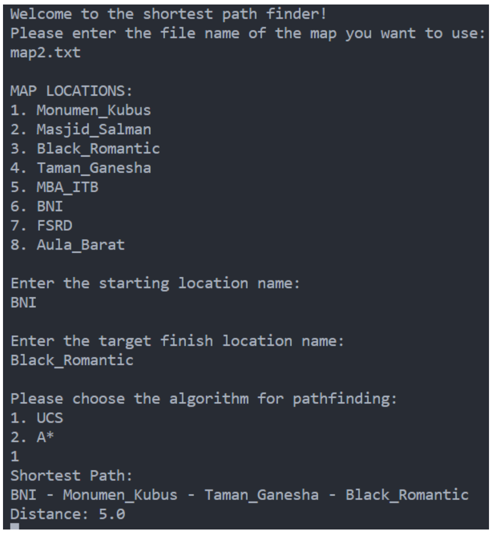
2.4.1 Peta di Kawasan ITB Ganesha

Pada folder test, peta ini diberi nama file map2.txt, dengan visualisasi dalam bentuk graf sebagai berikut,



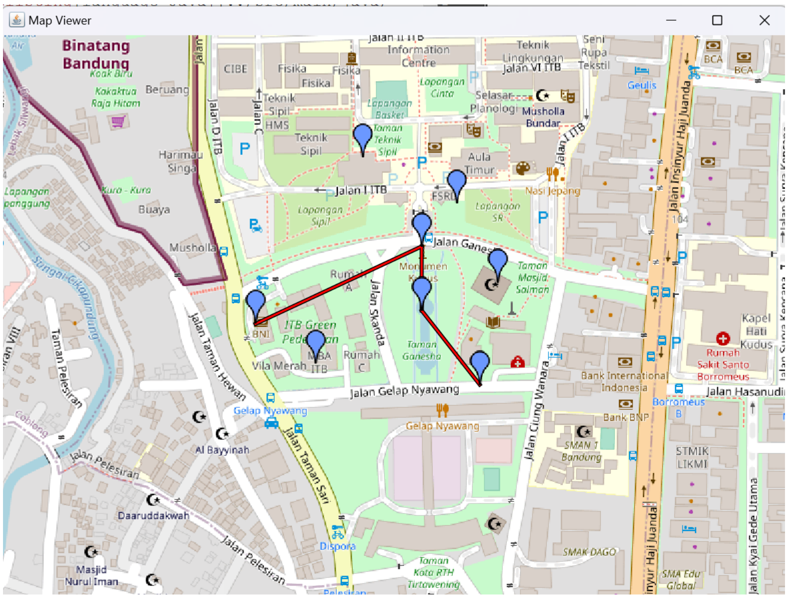
Gambar 2.1: Visualisasi Graf Map2.txt

Rute dari BNI - Black Romantic

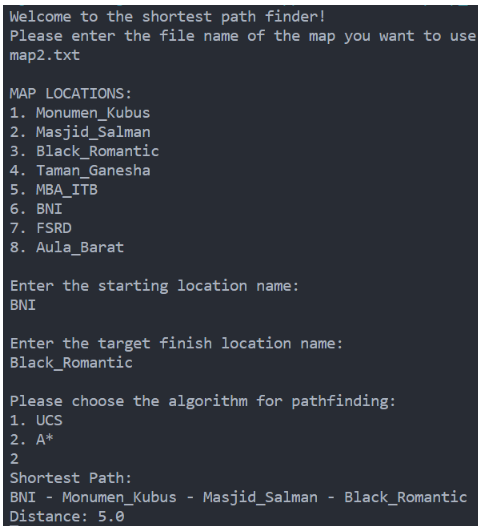


```
Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map2.txt

MAP LOCATIONS:
1. Monumen_Kubus
2. Masjid_Salman
3. Black_Romantic
4. Taman_Ganesha
5. MBA_ITB
6. BNI
7. FSRD
8. Aula_Barat

Enter the starting location name:
BNI

Enter the target finish location name:
Black_Romantic

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1
Shortest Path:
BNI - Monumen_Kubus - Taman_Ganesha - Black_Romantic
Distance: 5.0
```

Gambar 2.2: Pencarian dengan UCS.txt

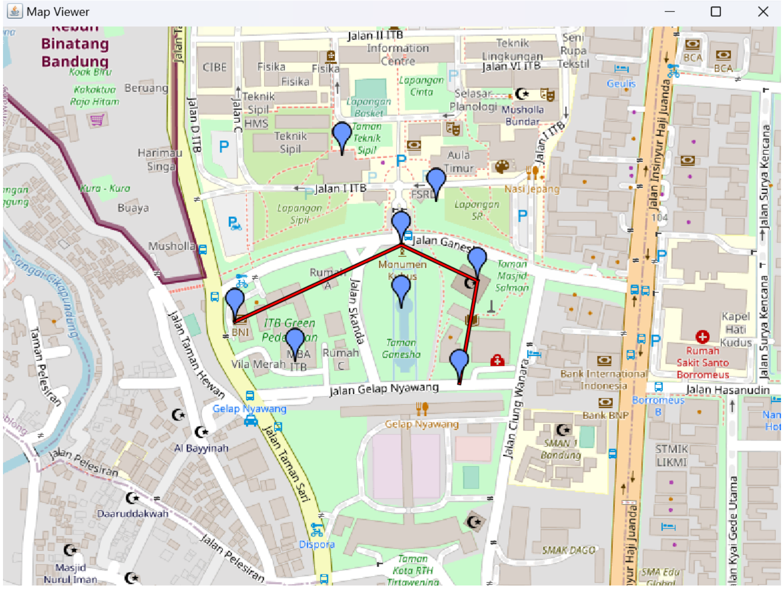


Gambar 2.3: Visualisasi Pencarian dengan UCS.txt



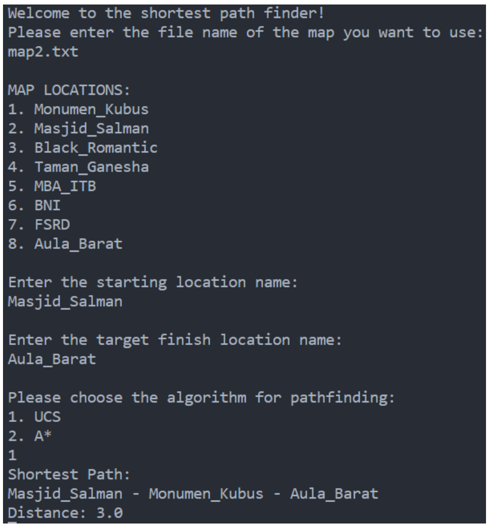
```
Welcome to the shortest path finder!
Please enter the file name of the map you want to use
map2.txt

MAP LOCATIONS:
1. Monumen_Kubus
2. Masjid_Salman
3. Black_Romantic
4. Taman_Ganesha
5. MBA_ITB
6. BNI
7. FSRD
8. Aula_Barat

Enter the starting location name:
BNI

Enter the target finish location name:
Black_Romantic

Please choose the algorithm for pathfinding:
1. UCS
2. A*
2
Shortest Path:
BNI - Monumen_Kubus - Masjid_Salman - Black_Romantic
Distance: 5.0
```

Gambar 2.4: Pencarian dengan A*.txt



Gambar 2.5: Visualisasi Pencarian dengan A*.txt

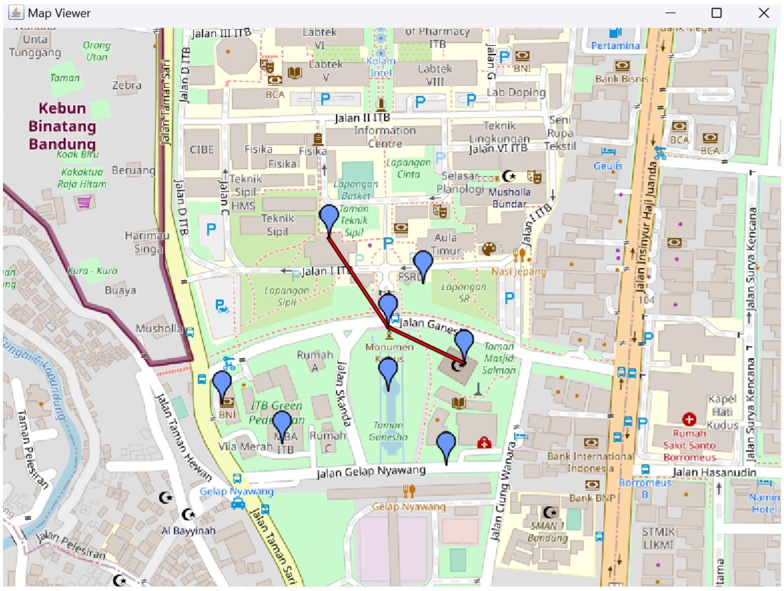
Rute dari Masjid Salman - Aula Barat



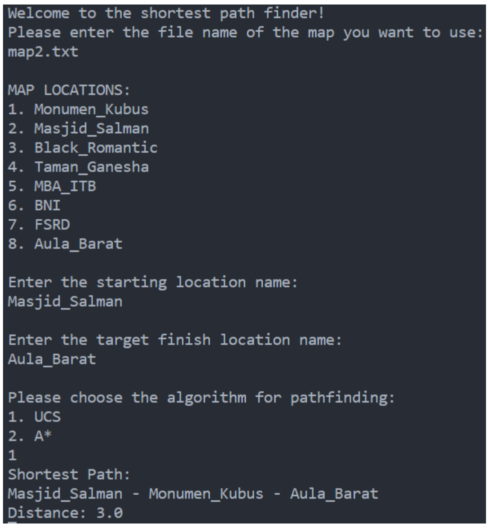
```
Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map2.txt

MAP LOCATIONS:
1. Monumen_Kubus
2. Masjid_Salman
3. Black_Romantic
4. Taman_Ganesha
5. MBA_ITB
6. BNI
7. FSRD
8. Aula_Barat

Enter the starting location name:
Masjid_Salman

Enter the target finish location name:
Aula_Barat

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1
Shortest Path:
Masjid_Salman - Monumen_Kubus - Aula_Barat
Distance: 3.0
```

Gambar 2.6: Pencarian dengan UCS.txt

2.4.2 Peta di Jakarta

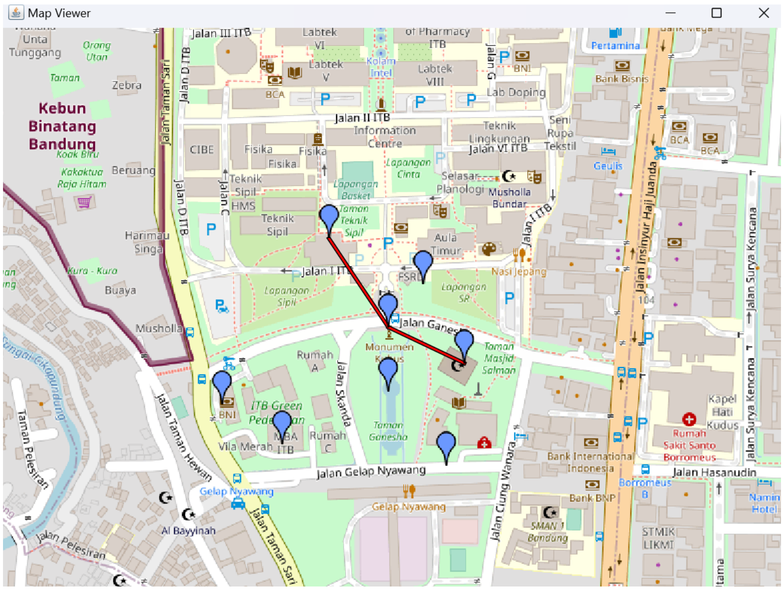


Gambar 2.7: Visualisasi Pencarian dengan UCS.txt



```
Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map2.txt

MAP LOCATIONS:
1. Monumen_Kubus
2. Masjid_Salman
3. Black_Romantic
4. Taman_Ganesha
5. MBA_ITB
6. BNI
7. FSRD
8. Aula_Barat

Enter the starting location name:
Masjid_Salman

Enter the target finish location name:
Aula_Barat

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1
Shortest Path:
Masjid_Salman - Monumen_Kubus - Aula_Barat
Distance: 3.0
```

Gambar 2.8: Pencarian dengan A*.txt



Gambar 2.9: Visualisasi Pencarian dengan A*.txt