

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

1	Latar Belakang	2
1.1	Deskripsi Persoalan	2
1.2	Algoritma Uniformed Cost Search (UCS)	2
1.3	Algoritma A*	2
2	Hasil	3
2.1	Penerapan UCS	3
2.2	Penerapan A*	3
2.3	Source Code	3
2.3.1	App.java	4
2.3.2	Solver.java	7
2.3.3	Node.java	10
2.3.4	Graph.java	11
2.3.5	Location.java	13
2.4	Hasil Pengujian	14
2.4.1	Peta di Kawasan ITB Ganesha	14
2.4.2	Peta di Kawasan Alun-Alun Bandung	18
2.4.3	Peta di Kawasan Buah Batu	21
2.4.4	Peta di Jakarta	23
2.4.5	Peta di Malang	26
3	Simpulan	28
3.1	Simpulan	28
3.2	Komentar	28
A	Pranala Github	29
B	Checklist	30

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

```

1  package stima;
2
3  import java.util.ArrayList;
4  import java.util.HashSet;
5  import java.util.List;
6  import java.util.Set;
7
8  import javax.swing.JFrame;
9
10 import org.jxmapviewer.JXMapView;
11 import org.jxmapviewer.OSMTileFactoryInfo;
12 import org.jxmapviewer.painter.CompoundPainter;
13 import org.jxmapviewer.painter.Painter;
14 import org.jxmapviewer.viewer.DefaultTileFactory;
15 import org.jxmapviewer.viewer.DefaultWaypoint;
16 import org.jxmapviewer.viewer.GeoPosition;
17 import org.jxmapviewer.viewer.TileFactoryInfo;
18 import org.jxmapviewer.viewer.Waypoint;
19 import org.jxmapviewer.viewer.WaypointPainter;
20
21 import algorithms.*;
22 import visuals.RoutePainter;
23
24 /**
25  * Aplikasi yang menerima file input graf map
26  * dan menampilkan hasil path terpendek dari point start ke
27  * point finish
28  * @author Matthew Mahendra
29  * @author Christophorus Dharma Winata
30  */
31 public class App
32 {
33     /**
34      * @param args the program args (ignored)
35      */
36     public static void main(String[] args)
37     {
38         // Opening java terminal
39         System.out.println("Welcome to the shortest path finder!");
40         System.out.println("Please enter the file name of the map you want to use:");
41         // Input file name
42         String fileName = System.console().readLine();
43         // Read file and create graph

```

```

44     Graph graph = new Graph(fileName);
45     // Print location
46     System.out.println("\nMAP LOCATIONS: ");
47     for (int i = 0; i < graph.getLocCount(); i++) {
48         System.out.println((i+1) + ". " + graph.getLocName(i)
49             );
50     }
51     //input start and finish location
52     System.out.println("\nEnter the starting location name:");
53     String startingPosition = System.console().readLine();
54     System.out.println("\nEnter the target finish location
55         name:");
56     String finishPosition = System.console().readLine();
57     // Calling solver from algorithms
58     Solver _solver = new Solver(startingPosition,
59         finishPosition, fileName);
60     // Choosing algorithm
61     System.out.println("\nPlease choose the algorithm for
62         pathfinding:");
63     System.out.println("1. UCS");
64     System.out.println("2. A*");
65     int choice = Integer.parseInt(System.console().readLine
66         ());
67     ArrayList<String> path;
68     if (choice == 1) {
69         // Path and distance from UCS algorithm
70         path = _solver.UCS();
71     } else if (choice == 2) {
72         // Path and distance from A* algorithm
73         path = _solver.AStar();
74     } else {
75         System.out.println("Invalid choice");
76         return;
77     }
78     System.out.println("\nShortest Path:");
79     for(int i = 0; i < path.size(); i++){
80         if(i != path.size() - 1){
81             System.out.print(path.get(i) + " - ");
82         }else{
83             System.out.println(path.get(i));
84         }
85     }
86     System.out.println("Distance: " + _solver.getJarak());

```

```

86
87 // Instantiate JXMapView
88 JXMapView mapView = new JXMapView();
89
90 // Display the viewer in a JFrame
91 JFrame frame = new JFrame("Map Viewer");
92 frame.getContentPane().add(mapView);
93 frame.setSize(800, 600);
94 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
95 frame.setVisible(true);
96
97 // Create a TileFactoryInfo for OpenStreetMap
98 TileFactoryInfo info = new OSMTileFactoryInfo();
99 DefaultTileFactory tileFactory = new DefaultTileFactory(
    info);
100 mapView.setTileFactory(tileFactory);
101
102 // Instantiating locations from input file
103 GeoPosition[] locationsOnMap = new GeoPosition[graph.
    getLocCount()];
104 for (int i = 0; i < graph.getLocCount(); i++) {
105     locationsOnMap[i] = new GeoPosition(graph.getPos(i));
106 }
107
108 // Create a track from the geo-positions
109 List<GeoPosition> solvedPath = new ArrayList<GeoPosition>
    >();
110
111 for (int i = 0; i < path.size(); i++) {
112     solvedPath.add(new GeoPosition(graph.getPos(path.get
        (i))));
113 }
114 // Calling RoutePainter from visuals
115 RoutePainter routePainter = new RoutePainter(solvedPath)
    ;
116
117 // Set the focus
118 double frac = 0.1;
119 if (frac * path.size() <= 1) {
120     mapView.zoomToBestFit(new HashSet<GeoPosition>(
        solvedPath), frac * path.size());
121 } else {
122     mapView.zoomToBestFit(new HashSet<GeoPosition>(
        solvedPath), 0.2);
123 }
124
125 // Create waypoints from the geo-positions

```

```

126     List<Waypoint> waypointsList = new ArrayList<Waypoint>()
127         ;
128     for (int i = 0; i < graph.getLocCount(); i++) {
129         waypointsList.add(new DefaultWaypoint(locationsOnMap
130             [i]));
131     }
132     Set<Waypoint> waypointsSet = new HashSet<Waypoint>(
133         waypointsList);
134
135     // Create a waypoint painter that takes all the
136     // waypoints
137     WaypointPainter<Waypoint> waypointPainter = new
138         WaypointPainter<Waypoint>();
139     waypointPainter.setWaypoints(waypointsSet);
140
141     // Create a compound painter that uses both the route-
142     // painter and the waypoint-painter
143     List<Painter<JXMapView>> painters = new ArrayList<
144         Painter<JXMapView>>();
145     painters.add(routePainter);
146     painters.add(waypointPainter);
147
148     CompoundPainter<JXMapView> painter = new
149         CompoundPainter<JXMapView>(painters);
150     mapView.setOverlayPainter(painter);
151 }
152 }

```

2.3.2 Solver.java

```

1 package algorithms;
2 import java.util.*;
3
4 public class Solver extends Graph{
5     private PriorityQueue<Node> queue;
6     private String startPoint, endPoint;
7     private double jarak;
8
9     public Solver(String sp, String ep, String fileName){
10         super(fileName);
11         startPoint = sp;
12         endPoint = ep;
13         queue = new PriorityQueue<>();
14         jarak = 0;
15     }
16
17     public ArrayList<String> AStar(){

```



```

18     queue = new PriorityQueue<>();
19
20     ArrayList<String> visit = new ArrayList<>();
21     visit.add(startPoint);
22     Node start = new Node(startPoint, endPoint, 0,
        euclideanDistance(getPos(startPoint), getPos(endPoint
        )), visit);
23     queue = new PriorityQueue<Node>();
24     queue.add(start);
25
26
27     while(queue.size() != 0){
28         Node check = queue.remove();
29
30         if(check.getCurrent().equals(check.getGoal())){
31             this.jarak = check.calculateFN();
32             return (check.getPath());
33         }
34
35         for(int i = 0; i < getNodes() ; i++){
36             if(getGraph(getIndex(check.getCurrent()), i) > 0
                && !check.getPath().contains(getLocName(i)))
            {
37                 /* Masukkan ke prioqueue
38                 * Buat nodes baru
39                 */
40                 ArrayList<String> visitNew = new ArrayList
                    <>(check.getPath());
41
42                 visitNew.add(getLocName(i));
43                 Node newNode = new Node(getLocName(i),
44                                         check.getGoal(),
45                                         getGraph(getIndex(
                                            check.getCurrent
                                                ()), i) + check.
                                                getGn(),
46                                         euclideanDistance(
                                            getPos(getLocName
                                                (i)), getPos(
                                            check.getGoal()))
47                                         ,
48                                         visitNew);
49                 queue.add(newNode);
50             }
51         }
52     }
53     return new ArrayList<>();

```

```

54
55     public ArrayList<String> UCS(){
56         queue = new PriorityQueue<>();
57
58         ArrayList<String> visitedLocs = new ArrayList<>();
59         visitedLocs.add(startPoint);
60         Node startNode = new Node(startPoint, endPoint, 0,
61             visitedLocs);
62         queue.add(startNode);
63
64         while(queue.size() != 0){
65             Node check = queue.remove();
66
67             if(check.getCurrent().equals(check.getGoal())){
68                 this.jarak = check.getGn();
69                 return (check.getPath());
70             }
71
72             for(int i = 0; i < getNodes(); i++){
73                 if(getGraph(getIndex(check.getCurrent()), i) > 0
74                     && !check.getPath().contains(getLocName(i)))
75                 {
76                     ArrayList<String> visitNew = new ArrayList
77                         <>(check.getPath());
78
79                     visitNew.add(getLocName(i));
80
81                     Node newNode = new Node(getLocName(i),
82                         check.getGoal(),
83                         getGraph(getIndex(
84                             check.getCurrent
85                                 ()), i) + check.
86                             getGn(),
87                         visitNew);
88
89                     queue.add(newNode);
90                 }
91             }
92         }
93
94         return new ArrayList<>();
95     }
96
97     public double getJarak(){
98         return jarak;
99     }
100 }

```

2.3.3 Node.java

```
1 package algorithms;
2 import java.util.*;
3
4 public class Node implements Comparable<Node>{
5     private String current;
6     private String goal;
7     private double gn;
8     private double hn;
9     private ArrayList<String> visited = new ArrayList<>();
10
11     /* Node untuk A* */
12     public Node(String c, String g, double gn, double hn,
13         ArrayList<String> visited){
14         current=c;
15         goal = g;
16         this.gn = gn;
17         this.hn = hn;
18         this.visited = visited;
19     }
20
21     /* Node untuk UCS, tidak ada nilai h(n) */
22     public Node(String c, String g, double gn, ArrayList<String>
23         visited){
24         current=c;
25         goal = g;
26         this.gn = gn;
27         this.hn = 0;
28         this.visited = visited;
29     }
30
31     public String getCurrent(){
32         return current;
33     }
34
35     public String getGoal(){
36         return goal;
37     }
38
39     public double getGn(){
40         return gn;
41     }
42
43     public double getHn(){
44         return hn;
45     }
46 }
```

```
45     public double calculateFN() {
46         return hn+gn;
47     }
48
49     public ArrayList<String> getPath() {
50         return visited;
51     }
52
53     @Override
54     public int compareTo(Node o) {
55         if (calculateFN() < o.calculateFN()) {
56             return -1;
57         } else if (calculateFN() == o.calculateFN()) {
58             return 0;
59         } else {
60             return 1;
61         }
62     }
63 }
```

2.3.4 Graph.java

```
1 package algorithms;
2 import java.io.File;
3 import java.io.FileNotFoundException;
4 import java.util.Scanner;
5
6 public class Graph {
7     protected int [][] graph;
8     protected int nodes;
9     protected Location [] loc;
10
11     public Graph(String filename){
12         try{
13             File file = new File("./test/", filename);
14             Scanner reader = new Scanner(file);
15
16             /* Ambil jumlah nodes dan set ukuran matrix */
17             String nString = reader.nextLine();
18             int n = Integer.parseInt(nString);
19             nodes = n;
20             graph = new int [n][n];
21
22             /* Insert lokasi */
23             loc = new Location[n];
24             for(int i = 0; i < n ; i++){
25                 String line = reader.nextLine();
```

```

26         String [] parse = line.split("\\s+");
27         loc[i] = new Location(parse[0], Double.
                parseDouble(parse[1]), Double.parseDouble(
                parse[2]));
28     }
29
30     /* Fill the matrix */
31     for(int i = 0; i < n ; i++){
32         String line = reader.nextLine();
33         String [] splited = line.split("\\s+");
34         for(int j = 0; j < n; j++){
35             graph[i][j] = Integer.parseInt(splited[j]);
36         }
37     }
38
39     reader.close();
40
41     }catch (FileNotFoundException e){
42         System.out.println("File not found!");
43         e.printStackTrace();
44     }
45 }
46 public double euclideanDistance(double[] l1, double[] l2){
47     return (Math.sqrt( Math.pow(l1[0]-l2[0], 2) + Math.pow(
        l1[1]-l2[1], 2) ));
48 }
49
50 public double[] getPos(String locName){
51     int idx = 0;
52     for(int i = 0; i < nodes; i++){
53         if(loc[i].getLocName().equals(locName))
54         {
55             idx = i;
56             break;
57         }
58     }
59     return loc[idx].getCoord();
60 }
61
62 /**
63  * Mengembalikan koordinat lokasi berdasarkan indeks
64  * @param i indeks lokasi
65  * @return double[] koordinat lokasi
66  */
67 public double[] getPos(int i){
68     return loc[i].getCoord();
69 }
70

```

```
71     public String getLocName(int i){
72         return loc[i].getLocName();
73     }
74
75     public int getIndex(String locName){
76         int idx = 0;
77         for(int i = 0; i < nodes; i++){
78             if(loc[i].getLocName().equals(locName))
79                 {
80                     idx = i;
81                     break;
82                 }
83         }
84         return idx;
85     }
86
87     public int getGraph(int b, int c){
88         return graph[b][c];
89     }
90
91     public int getNodes(){
92         return nodes;
93     }
94     public int getLocCount(){
95         return loc.length;
96     }
97 }
```

2.3.5 Location.java

```
1 package algorithms;
2 public class Location {
3     private String locName;
4     private double x;
5     private double y;
6
7     public Location(String locName, double x, double y){
8         this.x = x;
9         this.y = y;
10        this.locName = locName;
11    }
12
13    public String getLocName(){
14        return locName;
15    }
16
17    public double[] getCoord(){
```

```

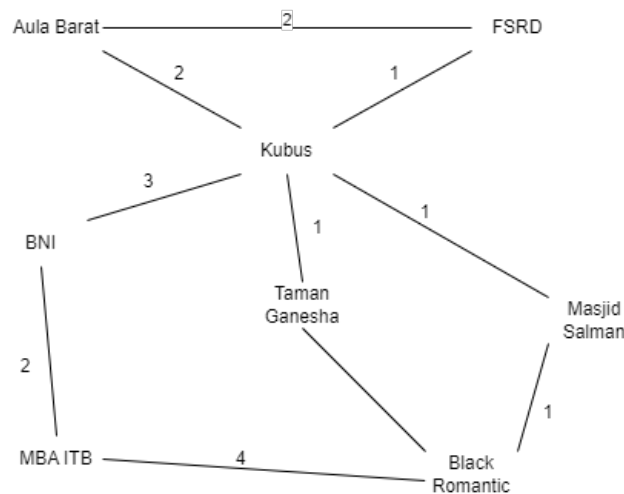
18     double[] coord = {this.x, this.y};
19     return coord;
20 }
21 }

```

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

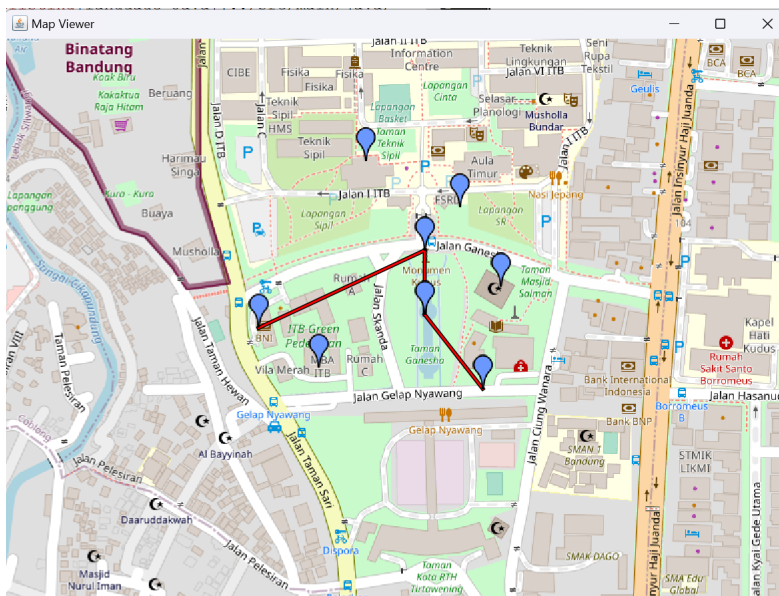
Enter the target finish location name:
Black_Romantic

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



Gambar 2.3: Visualisasi Pencarian dengan UCS

```

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

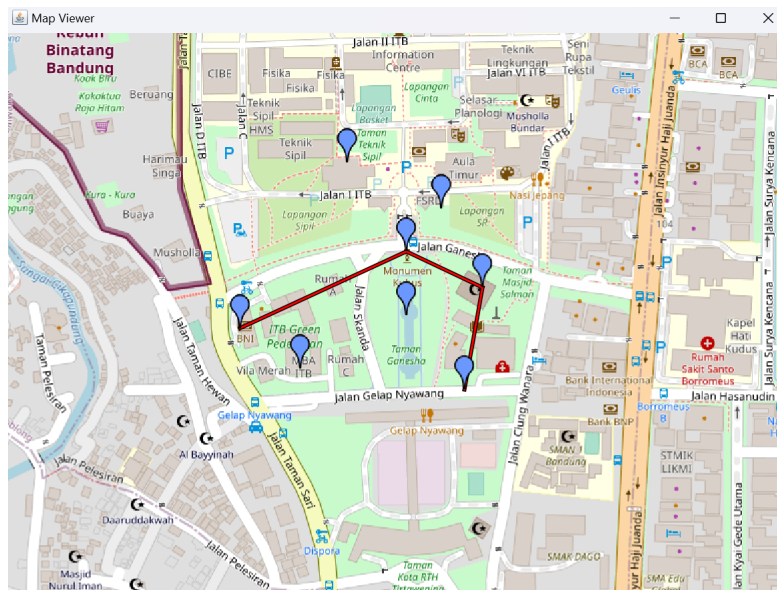
Enter the target finish location name:
Black_Romantic

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*



Gambar 2.5: Visualisasi Pencarian dengan A*

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

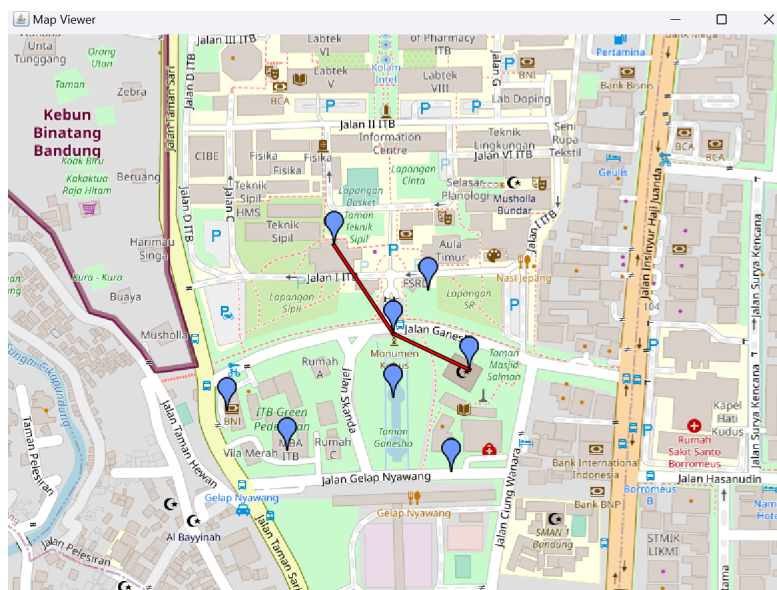
Enter the starting location name:
Masjid_Salman

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



Gambar 2.7: Visualisasi Pencarian dengan UCS

```

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

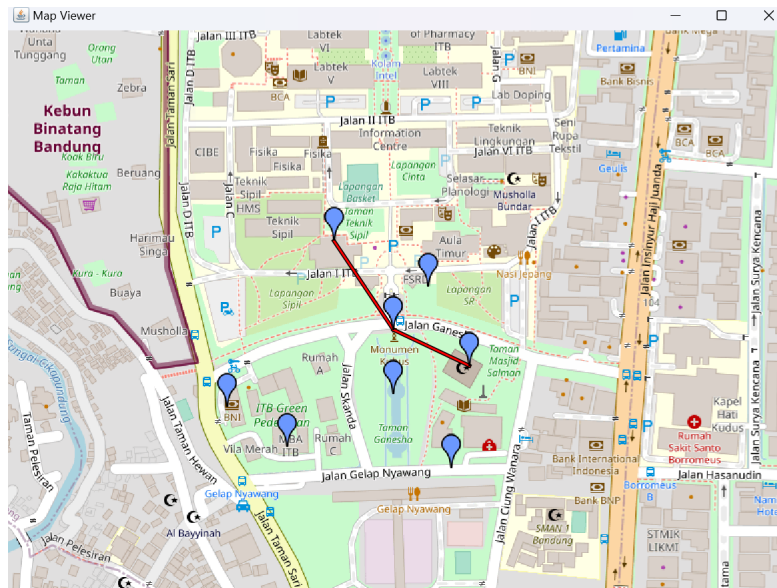
Enter the starting location name:
Masjid_Salman

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

```

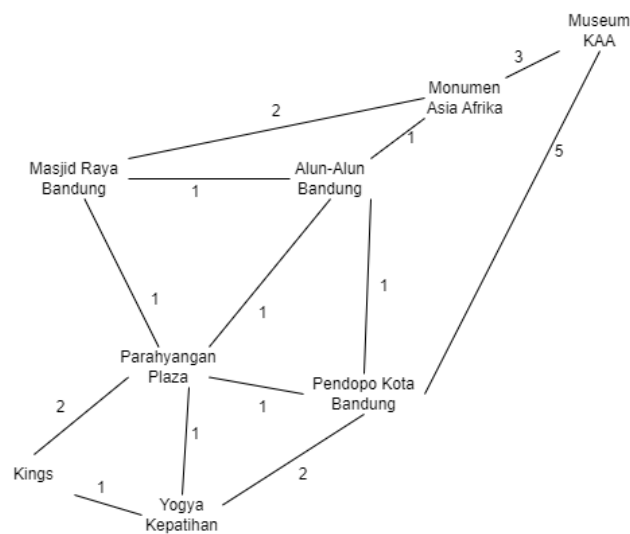
Gambar 2.8: Pencarian dengan A*



Gambar 2.9: Visualisasi Pencarian dengan A*

2.4.2 Peta di Kawasan Alun-Alun Bandung

Pada folder test, peta ini diberi nama file map4.txt, dengan visualisasi dalam bentuk graf sebagai berikut,

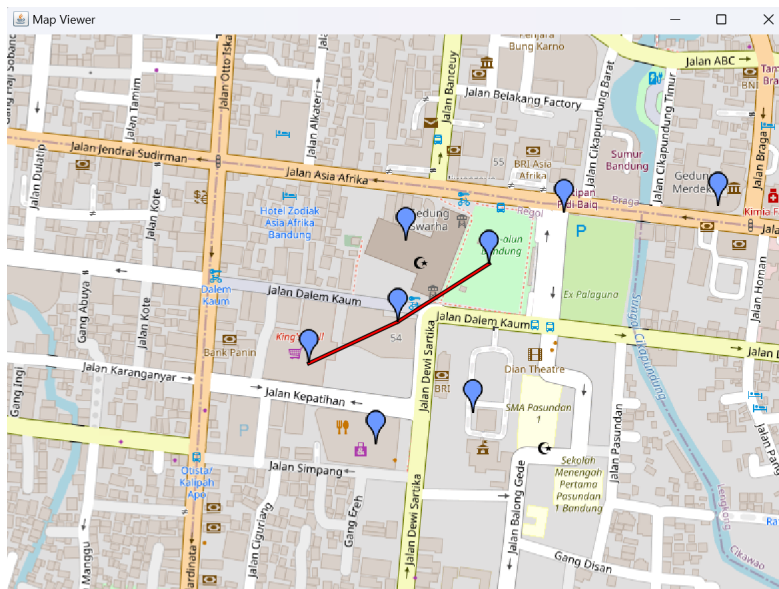


Gambar 2.10: Visualisasi Graf Map3.txt

Rute Alun-Alun Bandung - Kings

```
Welcome to the shortest path finder!  
Please enter the file name of the map you want to use:  
map3.txt  
  
MAP LOCATIONS:  
1. Alun-Alun_Bandung  
2. Masjid_Raya_Bandung  
3. Monumen_Asia_Afrika  
4. Parahyangan_Plaza  
5. Pendopo_Kota_Bandung  
6. Museum_KAA  
7. Yogya_Kepatihan  
8. Kings  
  
Enter the starting location name:  
Alun-Alun_Bandung  
  
Enter the target finish location name:  
Kings  
  
Please choose the algorithm for pathfinding:  
1. UCS  
2. A*  
1  
  
Shortest Path:  
Alun-Alun_Bandung - Parahyangan_Plaza - Kings  
Distance: 3.0
```

Gambar 2.11: Pencarian dengan UCS



Gambar 2.12: Visualisasi Pencarian dengan UCS

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map3.txt

MAP LOCATIONS:
1. Alun-Alun_Bandung
2. Masjid_Raya_Bandung
3. Monumen_Asia_Afrika
4. Parahyangan_Plaza
5. Pendopo_Kota_Bandung
6. Museum_KAA
7. Yogya_Kepatihan
8. Kings

Enter the starting location name:
Alun-Alun_Bandung

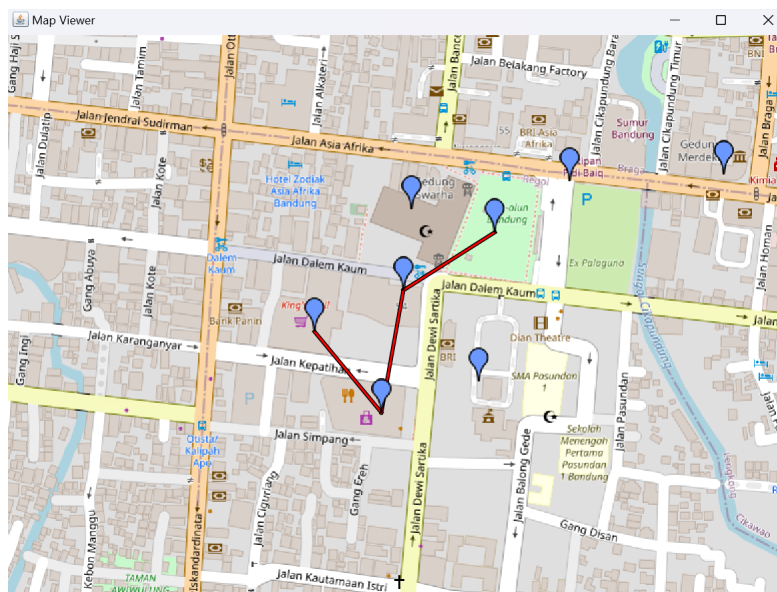
Enter the target finish location name:
Kings

Please choose the algorithm for pathfinding:
1. UCS
2. A*
2

Shortest Path:
Alun-Alun_Bandung - Parahyangan_Plaza - Yogya_Kepatihan - Kings
Distance: 3.0

```

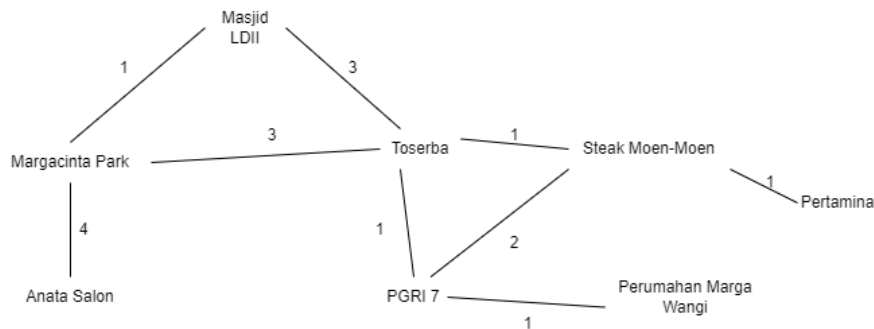
Gambar 2.13: Pencarian dengan A*



Gambar 2.14: Visualisasi Pencarian dengan A*

2.4.3 Peta di Kawasan Buah Batu

Pada folder test, peta ini diberi nama file map3.txt, dengan visualisasi dalam bentuk graf sebagai berikut,



Gambar 2.15: Visualisasi Graf Map4.txt

Perumahan Marga Wangi - Pertamina

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map4.txt

MAP LOCATIONS:
1. Marga_Cinta_Park
2. Anata_Salon
3. Masjid_LDII
4. Toserba
5. PGRI_7
6. Perumahan_Marga_Wangi
7. Pertamina
8. Steak_Moen_Moen

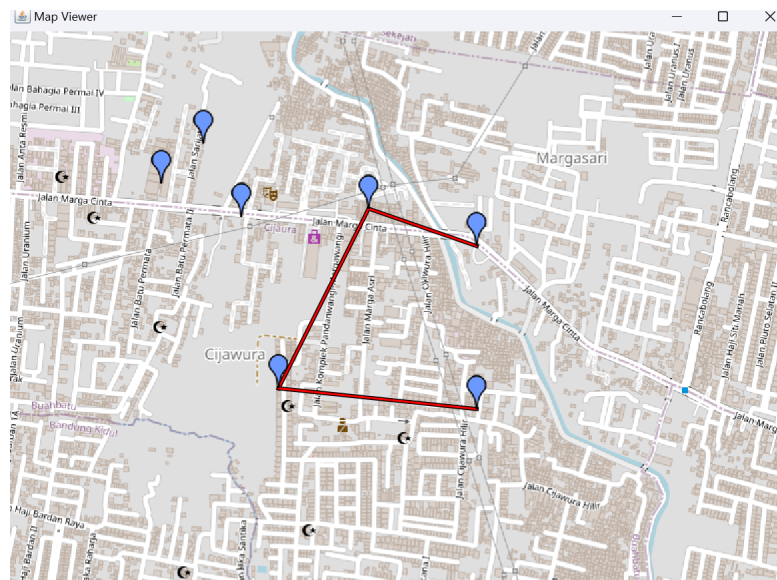
Enter the starting location name:
Perumahan_Marga_Wangi

Enter the target finish location name:
Pertamina

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1

Shortest Path:
Perumahan_Marga_Wangi - PGRI_7 - Steak_Moen_Moen - Pertamina
Distance: 4.0
  
```

Gambar 2.16: Pencarian dengan UCS



Gambar 2.17: Visualisasi Pencarian dengan UCS

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map4.txt

MAP LOCATIONS:
1. Marga_Cinta_Park
2. Anata_Salon
3. Masjid_LDII
4. Toserba
5. PGRI_7
6. Perumahan_Marga_Wangi
7. Pertamina
8. Steak_Moen_Moen

Enter the starting location name:
Perumahan_Marga_Wangi

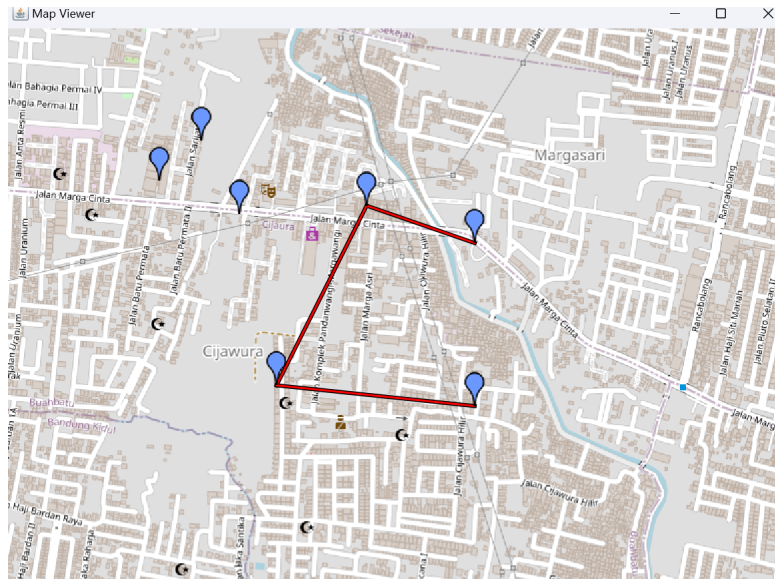
Enter the target finish location name:
Pertamina

Please choose the algorithm for pathfinding:
1. UCS
2. A*
2

Shortest Path:
Perumahan_Marga_Wangi - PGRI_7 - Steak_Moen_Moen - Pertamina
Distance: 4.0

```

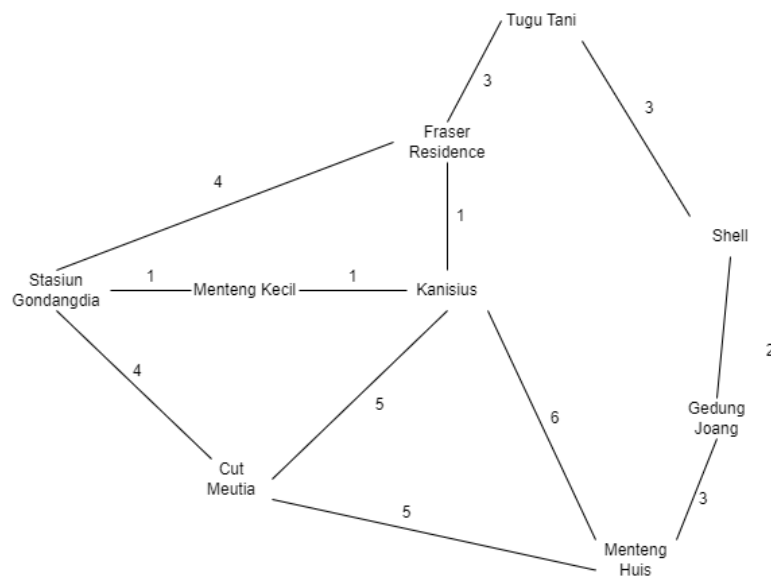
Gambar 2.18: Pencarian dengan A*



Gambar 2.19: Visualisasi Pencarian dengan A*

2.4.4 Peta di Jakarta

Pada folder test, peta ini diberi nama file map1.txt, dengan visualisasi dalam bentuk graf sebagai berikut,



Gambar 2.20: Visualisasi Graf Map1.txt

Rute Shell Menteng - Stasiun Gondangdia


```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
Map1.txt

MAP LOCATIONS:
1. Tugu_Tani
2. Kanisius
3. Masjid_Cut_Meutia
4. Menteng_Huis
5. Gedung_Joang
6. Shell_Menteng
7. Fraser_Residence
8. Stasiun_Gondangdia
9. Menteng_Kecil

Enter the starting location name:
Shell_Menteng

Enter the target finish location name:
Stasiun_Gondangdia

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1

Shortest Path:
Shell_Menteng - Tugu_Tani - Fraser_Residence - Kanisius - Menteng_Kecil - Stasiun_Gondangdia
Distance: 8.0

```

Gambar 2.21: Pencarian dengan UCS



Gambar 2.22: Visualisasi Pencarian dengan UCS

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map1.txt

MAP LOCATIONS:
1. Tugu_Tani
2. Kanisius
3. Masjid_Cut_Meutia
4. Menteng_Huis
5. Gedung_Joang
6. Shell_Menteng
7. Fraser_Residence
8. Stasiun_Gondangdia
9. Menteng_Kecil

Enter the starting location name:
Shell_Menteng

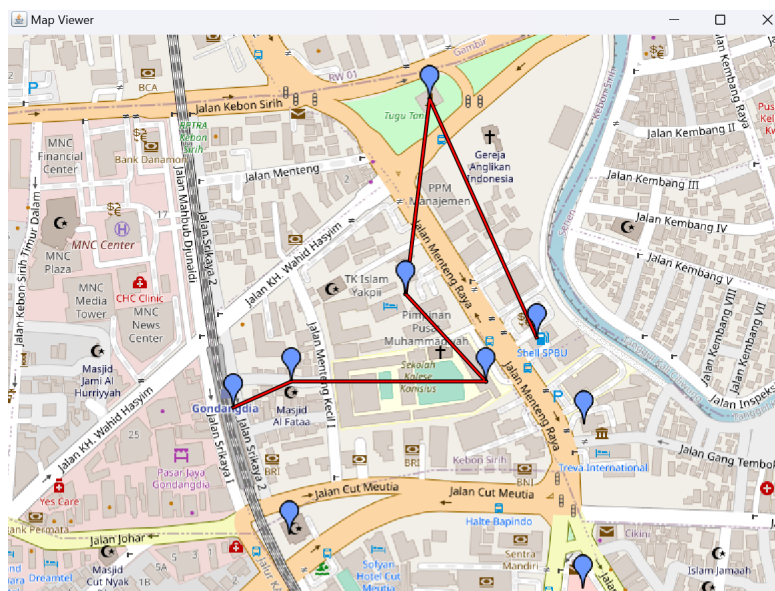
Enter the target finish location name:
Stasiun_Gondangdia

Please choose the algorithm for pathfinding:
1. UCS
2. A*
2

Shortest Path:
Shell_Menteng - Tugu_Tani - Fraser_Residence - Kanisius - Menteng_Kecil - Stasiun_Gondangdia
Distance: 8.0

```

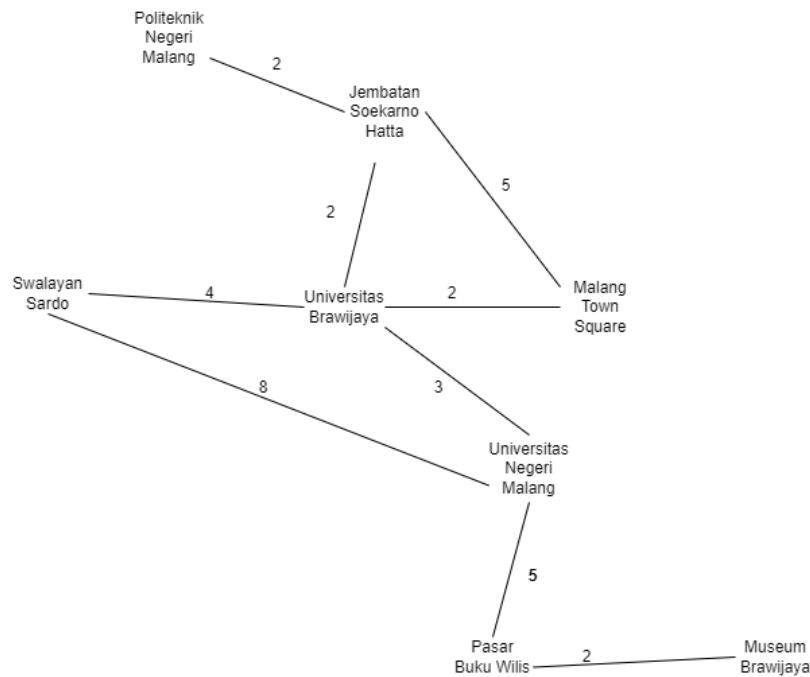
Gambar 2.23: Pencarian dengan A*



Gambar 2.24: Visualisasi Pencarian dengan A*

2.4.5 Peta di Malang

Pada folder test, peta ini diberi nama file map5.txt, dengan visualisasi dalam bentuk graf sebagai berikut,



Gambar 2.25: Visualisasi Graf Map5.txt

Rute Museum Brawijaya - Swalayan Sardo

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map5.txt

MAP LOCATIONS:
1. Universitas_Brawijaya
2. Universitas_Negeri_Malang
3. Swalayan_Sardo
4. Pasar_Buku_Wilis
5. Museum_Brawijaya
6. Jembatan_Soeta
7. Malang_Town_Square
8. Politeknik_Negeri_Malang

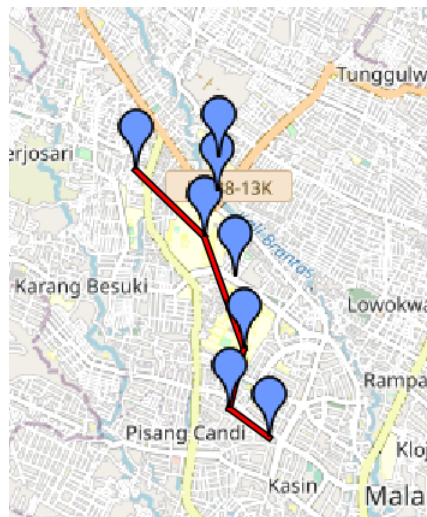
Enter the starting location name:
Museum_Brawijaya

Enter the target finish location name:
Swalayan_Sardo

Please choose the algorithm for pathfinding:
1. UCS
2. A*
1

Shortest Path:
Museum_Brawijaya - Pasar_Buku_Wilis - Universitas_Negeri_Malang - Universitas_Brawijaya - Swalayan_Sardo
Distance: 14.0
  
```

Gambar 2.26: Pencarian dengan UCS



Gambar 2.27: Visualisasi Pencarian dengan UCS

```

Welcome to the shortest path finder!
Please enter the file name of the map you want to use:
map5.txt

MAP LOCATIONS:
1. Universitas_Brawijaya
2. Universitas_Negeri_Malang
3. Swalayan_Sardo
4. Pasar_Buku_Wilis
5. Museum_Brawijaya
6. Jembatan_Soeta
7. Malang_Town_Square
8. Politeknik_Negeri_Malang

Enter the starting location name:
Museum_Brawijaya

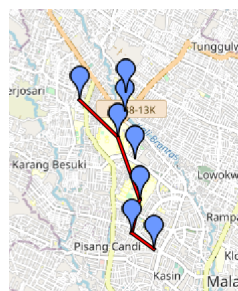
Enter the target finish location name:
Swalayan_Sardo

Please choose the algorithm for pathfinding:
1. UCS
2. A*
2

Shortest Path:
Museum_Brawijaya - Pasar_Buku_Wilis - Universitas_Negeri_Malang - Universitas_Brawijaya - Swalayan_Sardo
Distance: 14.0

```

Gambar 2.28: Pencarian dengan A*



Gambar 2.29: Visualisasi Pencarian dengan A*

BAB 3

Simpulan

3.1 Simpulan

Dari tugas kecil ini, dapat disimpulkan hal-hal sebagai berikut,

1. Algoritma pencarian rute terpendek adalah A* dan UCS. Keduanya merupakan modifikasi dari breadth-first search dengan menggunakan perhitungan
2. UCS merupakan uninformed search karena tidak ada perhitungan nilai heuristic dari suatu titik ke titik tujuan. UCS hanya mengandalkan nilai jarak dari suatu titik ke titik akhir, hingga ditemukan titik tujuan
3. A* merupakan informed search karena perhitungan nilai heuristic dari suatu titik ke titik tujuan membantu menentukan apakah suatu titik sudah lebih dekat ke titik yang ingin dituju tanpa perlu melakukan pergerakan yang tidak diperlukan

3.2 Komentar

- Matthew Mahendra: Jadi lebih paham UCS sama A* dibandingkan kalau cuman dengerin di kelas
- Christophorus Dharma Winata:

Lampiran A

Pranala Github

Tugas ini sudah dipublikasi pada Github dengan pranala https://github.com/MHEN2606/Tucil3_13521007_13521009

Lampiran B

Checklist

Poin	Ya	Tidak
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	✓	

Table B.1: Tabel Check List