

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	Perhitungan Jarak	3
2.4	Source Code	4
2.4.1	App.java	4
2.4.2	Solver.java	8
2.4.3	Node.java	10
2.4.4	Graph.java	12
2.4.5	Location.java	14
2.5	Hasil Pengujian	15
2.5.1	Peta di Kawasan ITB Ganesha	15
2.5.2	Peta di Kawasan Alun-Alun Bandung	17
2.5.3	Peta di Kawasan Buah Batu	19
2.5.4	Peta di Jakarta	21
2.5.5	Peta di Malang	24
3	Simpulan	26
3.1	Simpulan	26
3.2	Komentar	26
A	Pranala Github	27
B	Checklist	28

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 Perhitungan Jarak

Karena program dapat dikaitkan dengan peta asli, maka untuk perhitungan jarak menggunakan rumus Haversine untuk menentukan jarak aktual berdasarkan koordinat pada Bumi. Untuk kasus graf biasa, informasi jarak simpul ke simpul diberikan pada matriks ketetanggaan dan nilai heuristic dihitung menggunakan euclidean distance.

2.4 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. Graph.java berisi kelas untuk mendefinisikan graf. Location.java berisi kelas untuk mendefinisikan lokasi berikut koordinat. Node.java berisi kelas untuk mendefinisikan simpul yang menyimpan nilai $g(n)$ dan $h(n)$ dan digunakan pada saat menjalankan algoritma. Solver.java berisi kelas dengan metode algoritma UCS dan A*.

2.4.1 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 import java.awt.*;
8
9 import javax.swing.JFrame;
10
11 import org.jxmapviewer.JXMapView;
12 import org.jxmapviewer.OSMTileFactoryInfo;
13 import org.jxmapviewer.painter.CompoundPainter;
14 import org.jxmapviewer.painter.Painter;
15 import org.jxmapviewer.viewer.DefaultTileFactory;
16 import org.jxmapviewer.viewer.DefaultWaypoint;
17 import org.jxmapviewer.viewer.GeoPosition;
18 import org.jxmapviewer.viewer.TileFactoryInfo;
19 import org.jxmapviewer.viewer.Waypoint;
20 import org.jxmapviewer.viewer.WaypointPainter;
21
22 import algorithms.*;
23 import visuals.*;
24
25 /**
26  * Aplikasi yang menerima file input graf map
27  * dan menampilkan hasil path terpendek dari point start ke point
28  * finish
29  * @author Matthew Mahendra
30  * @author Christophorus Dharma Winata
31  */
32 public class App
33 {
34     /**
35      * @param args the program args (ignored)
36      */
37     public static void main(String[] args)
38     {
39         // Opening java terminal

```

```

39     System.out.println("Welcome to the shortest path finder!"
40     );
41     System.out.println("Please enter the file name of the map
42     you want to use:");
43     // Input file name
44     String fileName = System.console().readLine();
45     // Read file and create graph
46     Graph graph = new Graph(fileName);
47     // Print location
48     System.out.println("\nMAP LOCATIONS:");
49     for (int i = 0; i < graph.getLocCount(); i++) {
50         System.out.println((i+1) + ". " + graph.getLocName(i)
51         );
52     }
53     //input start and finish location
54     System.out.println("\nEnter the starting location name:");
55     ;
56     String startingPosition = System.console().readLine();
57     System.out.println("\nEnter the target finish location
58     name:");
59     String finishPosition = System.console().readLine();
60     // Calling solver from algorithms
61     Solver _solver = new Solver(startingPosition,
62     finishPosition, fileName);
63     // Choosing algorithm
64     System.out.println("\nPlease choose the algorithm for
65     pathfinding:");
66     System.out.println("1. UCS");
67     System.out.println("2. A*");
68     int choice = Integer.parseInt(System.console().readLine()
69     );
70     ArrayList<String> path;
71     if (choice == 1) {
72         // Path and distance from UCS algorithm
73         path = _solver.UCS();
74     } else if (choice == 2) {
75         // Path and distance from A* algorithm
76         path = _solver.AStar();
77     } else {
78         System.out.println("Invalid choice");
79         return;
80     }
81     System.out.println("\nShortest Path:");
82     for(int i = 0; i < path.size(); i++){
83         if(i != path.size() - 1){
84             System.out.print(path.get(i) + " - ");

```

```
82         }else{
83             System.out.println(path.get(i));
84         }
85     }
86
87     if(graph.isBonus()){
88         System.out.println("Distance:␣" + _solver.getJarak()
89             + "␣km");
90     }else{
91         System.out.println("Distance:␣" + _solver.getJarak())
92             ;
93     }
94
95     if(graph.isBonus()){
96         // Instantiate JXMapView
97         JXMapView mapView = new JXMapView();
98
99         // Display the viewer in a JFrame
100         JFrame frame = new JFrame("Map␣Viewer");
101         frame.getContentPane().add(mapView);
102         frame.setSize(800, 600);
103         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
104         frame.setVisible(true);
105
106         // Create a TileFactoryInfo for OpenStreetMap
107         TileFactoryInfo info = new OSMTileFactoryInfo();
108         DefaultTileFactory tileFactory = new
109             DefaultTileFactory(info);
110         mapView.setTileFactory(tileFactory);
111
112         // Instantiating locations from input file
113         GeoPosition[] locationsOnMap = new GeoPosition[graph.
114             getLocCount()];
115         for (int i = 0; i < graph.getLocCount(); i++) {
116             locationsOnMap[i]= new GeoPosition(graph.getPos(i
117                 ));
118         }
119
120         // Create a track from the geo-positions
121         List<GeoPosition> solvedPath = new ArrayList<
122             GeoPosition>();
123
124         for (int i = 0; i < path.size(); i++) {
125             solvedPath.add(new GeoPosition(graph.getPos(path.
126                 get(i))));
127         }
128
129         // Calling RoutePainter from visuals
130         RoutePainter routePainter = new RoutePainter(
131             solvedPath);
132
133         // Set the focus
```

```

125         double frac = 0.1;
126         if(frac * path.size() <= 1){
127             mapView.zoomToBestFit(new HashSet<GeoPosition>(
128                 solvedPath), frac*path.size());
129         }else{
130             mapView.zoomToBestFit(new HashSet<GeoPosition>(
131                 solvedPath), 0.2);
132         }
133         // Create waypoints from the geo-positions
134         List<Waypoint> waypointsList = new ArrayList<Waypoint>
135             >();
136         for (int i = 0; i < graph.getLocCount(); i++) {
137             waypointsList.add(new DefaultWaypoint(
138                 locationsOnMap[i]));
139         }
140         Set<Waypoint> waypointsSet = new HashSet<Waypoint>(
141             waypointsList);
142
143         // Create a waypoint painter that takes all the
144         // waypoints
145         WaypointPainter<Waypoint> waypointPainter = new
146             WaypointPainter<Waypoint>();
147         waypointPainter.setWaypoints(waypointsSet);
148
149         // Create a compound painter that uses both the route
150         // -painter and the waypoint-painter
151         List<Painter<JXMapView>> painters = new ArrayList<
152             Painter<JXMapView>>();
153         painters.add(routePainter);
154         painters.add(waypointPainter);
155
156         CompoundPainter<JXMapView> painter = new
157             CompoundPainter<JXMapView>(painters);
158         mapView.setOverlayPainter(painter);
159     }else{
160         JFrame frame = new JFrame("Graph_Viewer");
161         GraphPainter graphPainter = new GraphPainter(path,
162             graph);
163         graphPainter.setPreferredSize(new Dimension(600, 600)
164             );
165         frame.add(graphPainter);
166         frame.pack();
167         // frame.setLocationRelativeTo(null);
168         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
169         frame.setVisible(true);
170     }
171 }

```



```

46         check.getGoal(),
47         haversine(getPos(
            check.getCurrent()
            ), getPos(
                getLocName(i))) +
                check.getGn(),
48         haversine(getPos(
            getLocName(i)),
            getPos(check.
                getGoal()))),
49         visitNew);
50     }else{
51         newNode = new Node(getLocName(i),
52         check.getGoal(),
53         getGraph(getIndex(
            check.getCurrent()
            ), i) + check.
            getGn(),
54         euclideanDistance(
            getPos(getLocName(
                i)), getPos(check.
                getGoal()))),
55         visitNew);
56     }
57     queue.add(newNode);
58 }
59 }
60 }
61 return new ArrayList<>();
62 }
63
64 public ArrayList<String> UCS(){
65     queue = new PriorityQueue<>();
66
67     ArrayList<String> visitedLocs = new ArrayList<>();
68     visitedLocs.add(startPoint);
69     Node startNode = new Node(startPoint, endPoint, 0,
        visitedLocs);
70     queue.add(startNode);
71
72     while(queue.size() != 0){
73         Node check = queue.remove();
74
75         if(check.getCurrent().equals(check.getGoal())){
76             this.jarak = check.getGn();
77             return (check.getPath());
78         }
79
80         for(int i = 0; i < getNodes() ; i++){
81             if(getGraph(getIndex(check.getCurrent()), i) > 0
                && !check.getPath().contains(getLocName(i))){

```

```

82         ArrayList<String> visitNew = new ArrayList<>(
            check.getPath());
83
84         visitNew.add(getLocName(i));
85
86         Node newNode;
87
88         if(isBonus()){
89             newNode = new Node(getLocName(i),
90                               check.getGoal(),
91                               haversine(getPos(
92                                     check.getCurrent()
93                                     ), getPos(
94                                         getLocName(i))) +
95                               check.getGn(),
96                               visitNew);
97         }else{
98             newNode = new Node(getLocName(i),
99                               check.getGoal(),
100                              getGraph(getIndex(
101                                    check.getCurrent()
102                                    ), i) + check.
103                              getGn(),
104                              visitNew);
105         }
106
107         queue.add(newNode);
108     }
109 }
110
111     public double getJarak(){
112         return jarak;
113     }
114 }

```

2.4.3 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  /* Node untuk UCS, tidak ada nilai h(n) */
21  public Node(String c, String g, double gn, ArrayList<String>
22      visited){
23      current=c;
24      goal = g;
25      this.gn = gn;
26      this.hn = 0;
27      this.visited = visited;
28  }
29  public String getCurrent(){
30      return current;
31  }
32
33  public String getGoal(){
34      return goal;
35  }
36
37  public double getGn(){
38      return gn;
39  }
40
41  public double getHn(){
42      return hn;
43  }
44
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 }
```

2.4.4 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    protected boolean isBonus;
11
12    public Graph(String filename){
13        try{
14            File file = new File("./test/", filename);
15            Scanner reader = new Scanner(file);
16
17            /* Ambil jumlah nodes dan set ukuran matrix */
18            String nString = reader.nextLine();
19            int n = Integer.parseInt(nString);
20            nodes = n;
21            graph = new int[n][n];
22
23            /* Insert lokasi */
24            loc = new Location[n];
25            for(int i = 0; i < n ; i++){
26                String line = reader.nextLine();
27                String[] parse = line.split("\\s+");
28                loc[i] = new Location(parse[0], Double.
                    parseDouble(parse[1]), Double.parseDouble(
                        parse[2]));
29            }
30
31            /* Fill the matrix */
32            for(int i = 0; i < n ; i++){
33                String line = reader.nextLine();
34                String[] splited = line.split("\\s+");
35                for(int j = 0; j < n; j++){
36                    graph[i][j] = Integer.parseInt(splited[j]);
37                }
38            }
39
40            /* Penentuan Bonus atau bukan */
41            for(int i = 0; i < n ; i++){
```

```
42         for(int j = 0; j < n ; j++){
43             if(graph[i][j] > 1){
44                 isBonus = false;
45                 break;
46             }
47             isBonus = true;
48         }
49     }
50
51
52     reader.close();
53
54     }catch (FileNotFoundException e){
55         System.out.println("File not found!");
56         e.printStackTrace();
57     }
58 }
59 public double euclideanDistance(double[] l1, double[] l2){
60     return (Math.sqrt( Math.pow(l1[0]-l2[0], 2) + Math.pow(l1
61         [1]-l2[1], 2) ));
62 }
63 public double haversine(double[] l1, double[] l2){
64     double dLat = Math.toRadians(l2[0] - l1[0]);
65     double dLon = Math.toRadians(l2[1] - l1[1]);
66
67     double lat1 = Math.toRadians(l1[0]);
68     double lat2 = Math.toRadians(l2[0]);
69
70     double a = Math.pow(Math.sin(dLat / 2), 2) + Math.pow(
71         Math.sin(dLon / 2), 2) * Math.cos(lat1) * Math.cos(
72         lat2);
73
74     double rad = 6371;
75     double c = 2 * Math.asin(Math.sqrt(a));
76     return rad * c;
77 }
78 public double[] getPos(String locName){
79     int idx = 0;
80     for(int i = 0; i < nodes; i++){
81         if(loc[i].getLocName().equals(locName))
82         {
83             idx = i;
84             break;
85         }
86     }
87     return loc[idx].getCoord();
88 }
89 /**
```

```

90      * Mengembalikan koordinat lokasi berdasarkan indeks
91      * @param i indeks lokasi
92      * @return double[] koordinat lokasi
93      */
94      public double[] getPos(int i){
95          return loc[i].getCoord();
96      }
97
98      public String getLocName(int i){
99          return loc[i].getLocName();
100     }
101
102     public int getIndex(String locName){
103         int idx = 0;
104         for(int i = 0; i < nodes; i++){
105             if(loc[i].getLocName().equals(locName))
106             {
107                 idx = i;
108                 break;
109             }
110         }
111         return idx;
112     }
113
114     public int getGraph(int b, int c){
115         return graph[b][c];
116     }
117
118     public int getNodes(){
119         return nodes;
120     }
121     public int getLocCount(){
122         return loc.length;
123     }
124
125     public Location[] getLocation(){
126         return loc;
127     }
128
129     public boolean isBonus(){
130         return isBonus;
131     }
132 }

```

2.4.5 Location.java

```

1 package algorithms;
2 public class Location {
3     private String locName;
4     private double x;

```

```

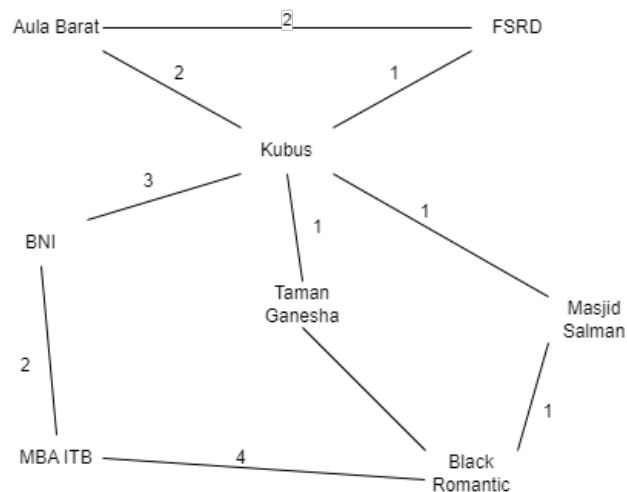
5     private double y;
6
7     public Location(String locName, double x, double y){
8         this.x = x;
9         this.y = y;
10        this.locName = locName;
11    }
12
13    public String getLocName(){
14        return locName;
15    }
16
17    public double[] getCoord(){
18        double[] coord = {this.x, this.y};
19        return coord;
20    }
21 }

```

2.5 Hasil Pengujian

2.5.1 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

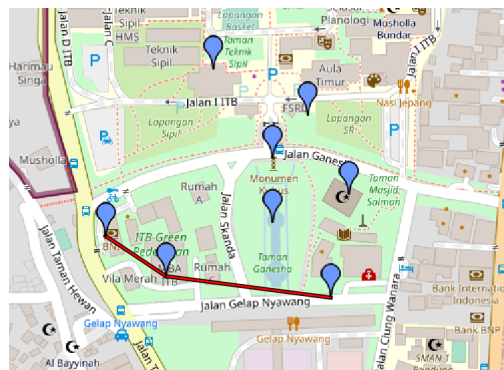
Enter the target finish location name:
Black_Romantic

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

Shortest Path:
BNI - MBA_ITB - Black_Romantic
Distance: 0.28114556844772376 km

```

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

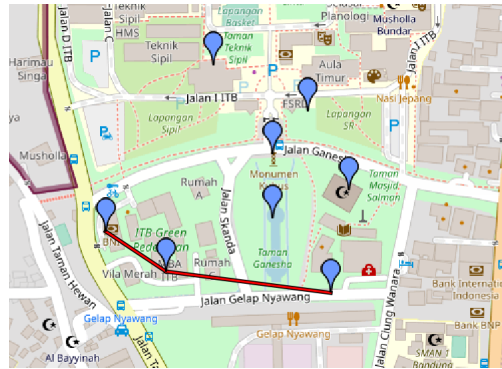
Enter the target finish location name:
Black_Romantic

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

Shortest Path:
BNI - MBA_ITB - Black_Romantic
Distance: 0.28114556844772376 km

```

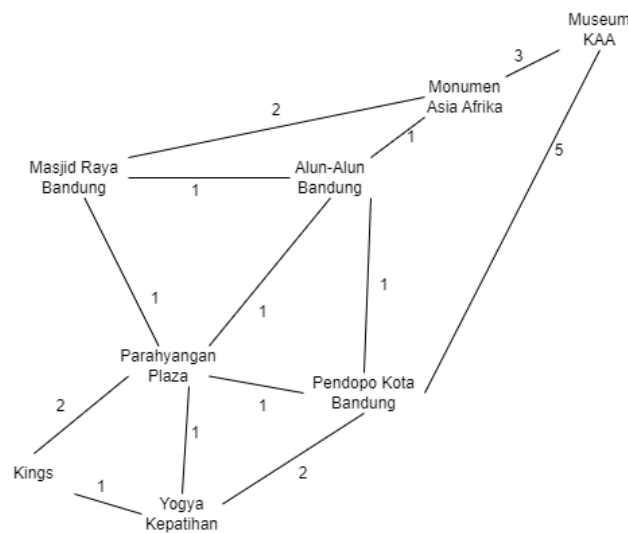
Gambar 2.4: Pencarian dengan A*



Gambar 2.5: Visualisasi Pencarian dengan A*

2.5.2 Peta di Kawasan Alun-Alun Bandung

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



Gambar 2.6: 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

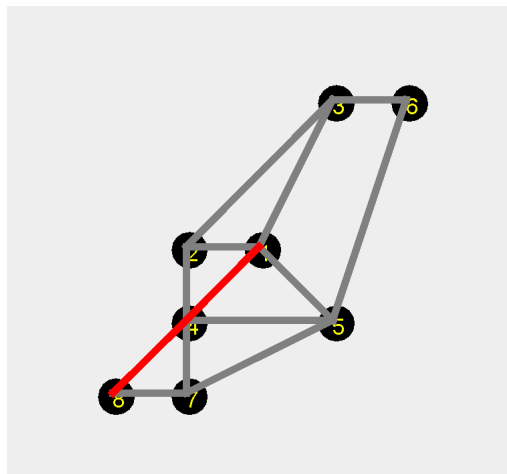
Enter the target finish location name:
Kings

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

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

```

Gambar 2.7: Pencarian dengan UCS



Gambar 2.8: 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

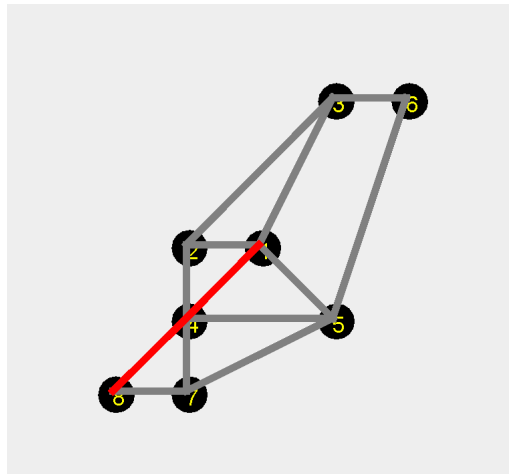
Enter the target finish location name:
Kings

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

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

```

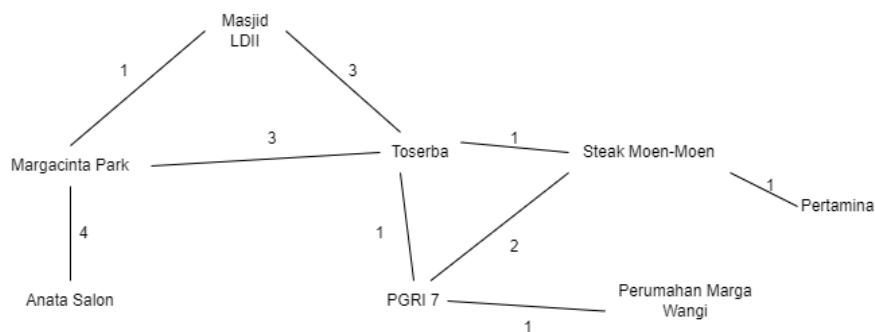
Gambar 2.9: Pencarian dengan A*



Gambar 2.10: Visualisasi Pencarian dengan A*

2.5.3 Peta di Kawasan Buah Batu

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



Gambar 2.11: 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

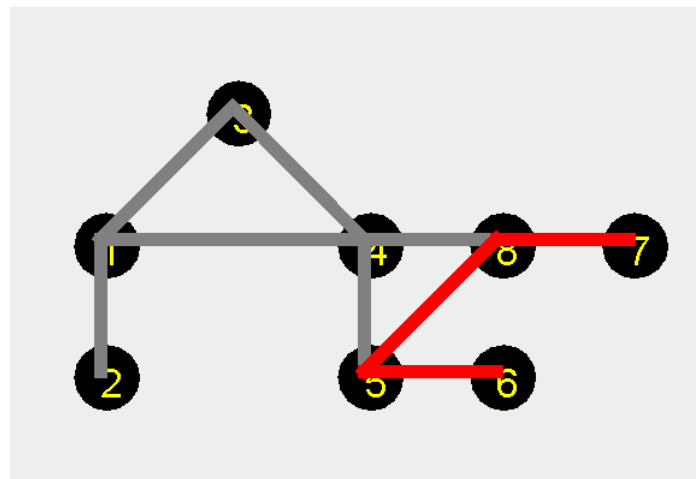
Enter the target finish location name:
Pertamina

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 2.12: Pencarian dengan UCS



Gambar 2.13: 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

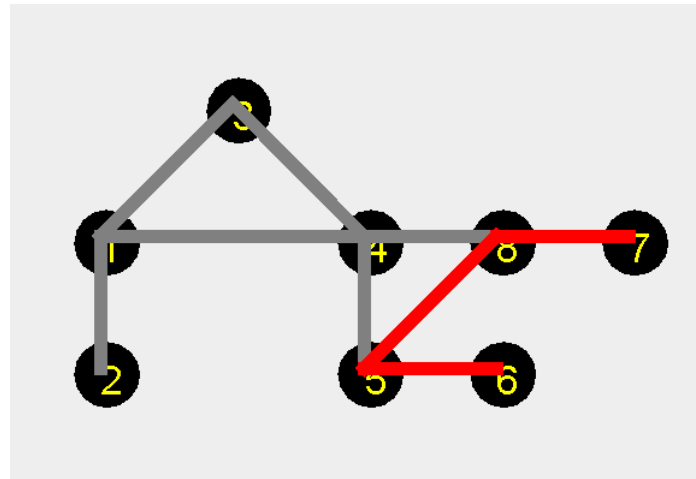
Enter the target finish location name:
Pertamina

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

```

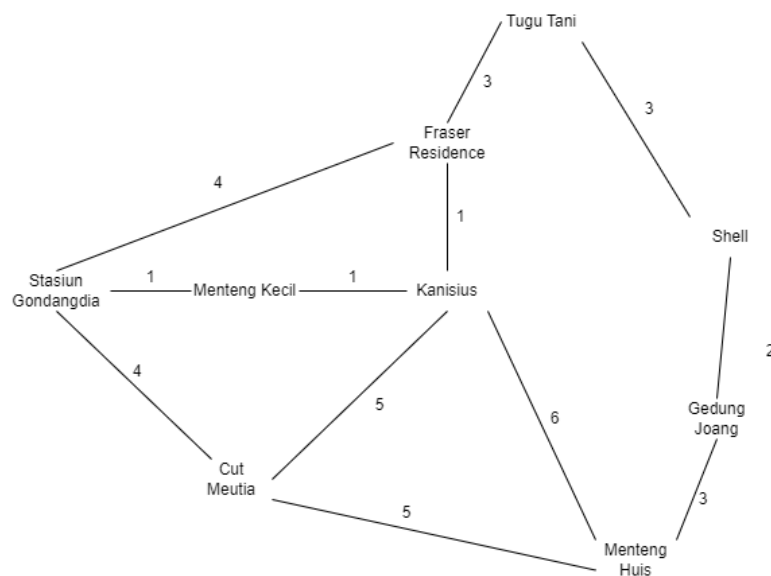
Gambar 2.14: Pencarian dengan A*



Gambar 2.15: Visualisasi Pencarian dengan A*

2.5.4 Peta di Jakarta

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

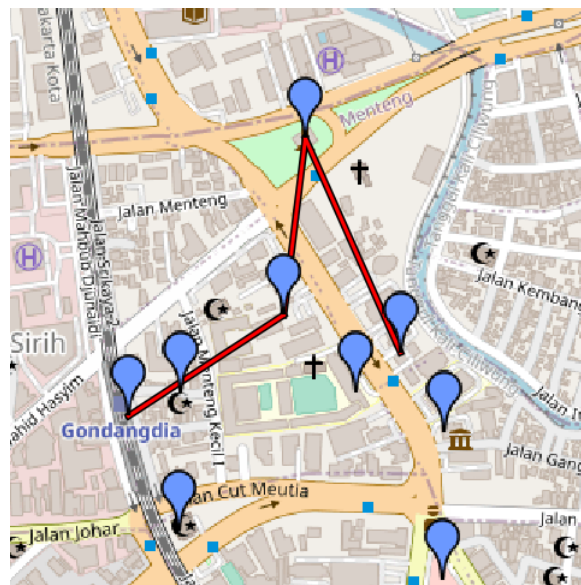


Gambar 2.16: Visualisasi Graf Map1.txt

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 - Stasiun_Gondangdia  
Distance: 0.7877664587110378 km
```

Gambar 2.17: Pencarian dengan UCS



Gambar 2.18: 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

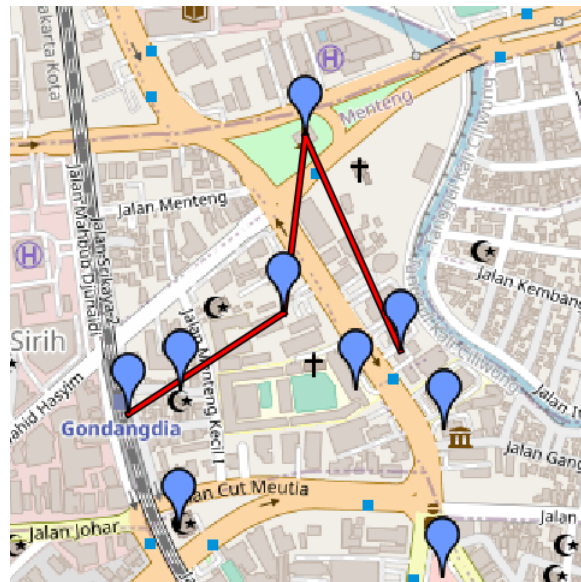
Enter the target finish location name:
Stasiun_Gondangdia

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

Shortest Path:
Shell_Menteng - Tugu_Tani - Fraser_Residence - Stasiun_Gondangdia
Distance: 0.7877664587110378 km

```

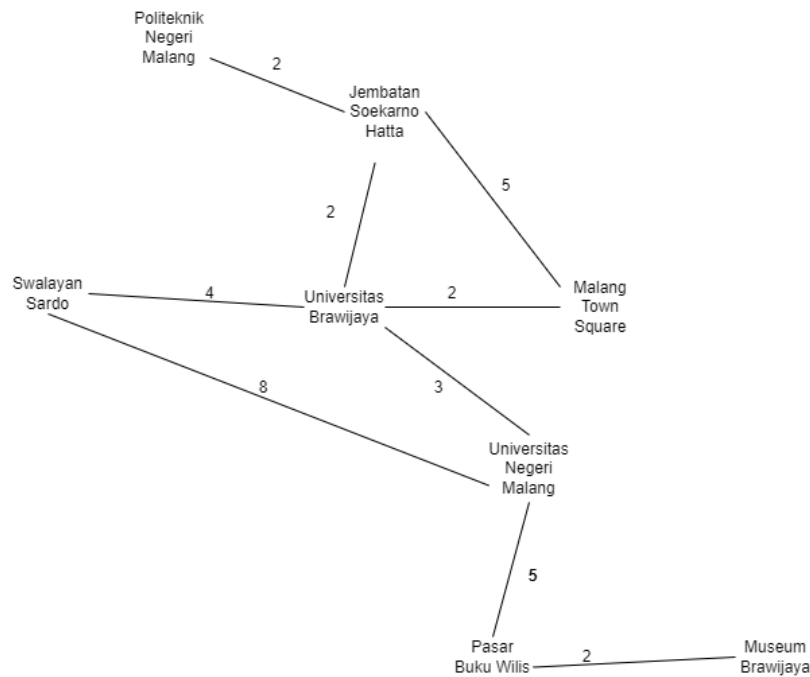
Gambar 2.19: Pencarian dengan A*



Gambar 2.20: Visualisasi Pencarian dengan A*

2.5.5 Peta di Malang

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



Gambar 2.21: 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

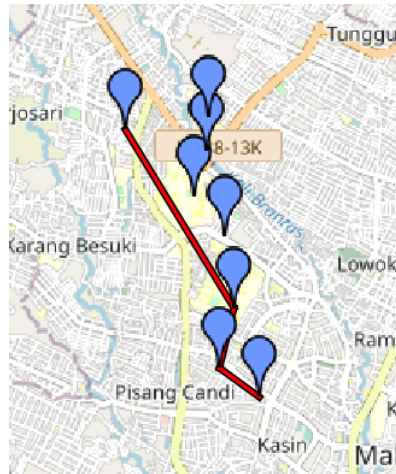
Enter the starting location name:
Museum_Brawijaya

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 - Swalayan_Sardo
Distance: 3.341713374511008 km
  
```

Gambar 2.22: Pencarian dengan UCS



Gambar 2.23: 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

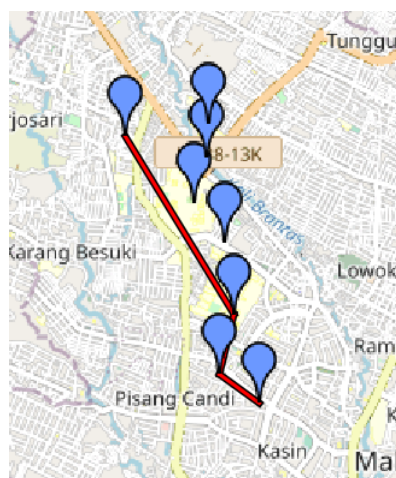
Enter the target finish location name:
Swalayan_Sardo

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

Shortest Path:
Museum_Brawijaya - Pasar_Buku_Wilis - Universitas_Negeri_Malang - Swalayan_Sardo
Distance: 3.341713374511008 km

```

Gambar 2.24: Pencarian dengan A*



Gambar 2.25: 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: Semangat kuliahnya! Tekuni kelas!

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