

IF2211 - Strategi Algoritma
Laporan Tugas Kecil 3



Disusun Oleh:

Haikal Ardzi Shofiyurrohman	13521012
Muhammad Zulfiansyah Bayu P.	13521028

Institut Teknologi Bandung
Sekolah Teknik Elektro dan Informatika
Tahun Ajaran 2022/2023

Daftar Isi

IF2211 - Strategi Algoritma	1
Laporan Tugas Kecil 3	1
Daftar Isi	2
1 Deskripsi Masalah	1
2 Kode Program	2
2.1 AStar.java	2
2.2 UCS.java	5
3 Eksperimen	7
4 Repository Github	11
5 Kesimpulan, Komentar, dll	12
6 Lampiran	13

1 Deskripsi Masalah

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.



Gambar 1.1. Google Map

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

2 Kode Program

2.1 AStar.java

```
package Algorithm;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.PriorityQueue;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JLabel;

public class AStar {

    private final double[][] adjacencyMatrix;
    private final int numberOfNodes;

    public AStar(double[][] adjacencyMatrix) {
        this.adjacencyMatrix = adjacencyMatrix;
        this.numberOfNodes = adjacencyMatrix.length;
    }

    public ArrayList<Integer> findShortestPath(int startNode, int endNode) {
        // Array untuk menyimpan jarak terpendek dari startNode ke
        // setiap node
        double[] shortestDistances = new double[numberOfNodes];
        Arrays.fill(shortestDistances, Integer.MAX_VALUE);
        shortestDistances[startNode] = 0;

        // Priority queue untuk menyimpan node-node yang akan
        // dieksplorasi
        PriorityQueue<Node> queue = new
PriorityQueue<>(Comparator.comparingDouble(node -> node.fScore));
        queue.add(new Node(startNode, 0, heuristic(startNode,
endNode), null));

        // Array untuk menyimpan node-node yang sudah dieksplorasi
        boolean[] visited = new boolean[numberOfNodes];
```

```

while (!queue.isEmpty()) {
    Node currentNode = queue.poll();
    int currentNodeIndex = currentNode.index;

    if (currentNodeIndex == endNode) {
        // Kita sudah mencapai node tujuan, maka kita dapat
        mengembalikan path terpendek dari startNode ke endNode
        ArrayList<Integer> path = new ArrayList<>();
        Node node = currentNode;
        while (node != null) {
            path.add(0, node.index);
            node = node.parent;
        }
        // Munculkan cost
        JPanel panel = new JPanel();
        panel.add(new JLabel("Cost : " +
        currentNode.gScore));
        JOptionPane.showMessageDialog(null, panel, "Cost
        (Algoritma A*)", JOptionPane.PLAIN_MESSAGE);

        return path;
    }

    if (visited[currentNodeIndex]) {
        // Node ini sudah dieksplorasi, lewati
        continue;
    }

    visited[currentNodeIndex] = true;

    for (int i = 0; i < numberOfNodes; i++) {
        if (adjacencyMatrix[currentNodeIndex][i] != 0 &&
        !visited[i]) {
            // Node tetangga yang belum dieksplorasi
            double tentativeDistance =
            shortestDistances[currentNodeIndex] +
            adjacencyMatrix[currentNodeIndex][i];
            if (tentativeDistance < shortestDistances[i]) {
                // Node tetangga dapat dicapai dengan jarak
                terpendek melalui currentNode
                shortestDistances[i] = tentativeDistance;
                Node neighborNode = new Node(i,

```

```

shortestDistances[i], heuristic(i, endNode), currentNode);
queue.add(neighborNode);
}
}
}
JPanel panel = new JPanel();
panel.add(new JLabel("Tidak ditemukan path!"));
 JOptionPane.showMessageDialog(null, panel, "Cost (Algoritma A*)",
 JOptionPane.PLAIN_MESSAGE);
// Tidak ditemukan path dari startNode ke endNode
return null;
}

private int heuristic(int nodeIndex, int endNodeIndex) {
// Fungsi heuristik, misalnya menggunakan jarak Euclidean
int x1 = nodeIndex / numberOfNodes;
int y1 = nodeIndex % numberOfNodes;
int x2 = endNodeIndex / numberOfNodes;
int y2 = endNodeIndex % numberOfNodes;
return (int) Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 -
y2, 2));
}

private static class Node {
public final int index;
public final double gScore;
public final double hScore;
public final double fScore;
public final Node parent;

public Node(int index, double gScore, double hScore, Node
parent) {
this.index = index;
this.gScore = gScore;
this.hScore = hScore;
this.fScore = gScore + hScore;
this.parent = parent;
}
}
}
}

```

2.2 Algorithm_UCS.java

```
package Algorithm;

import java.util.*;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JLabel;

public class UCS {

    private final double[][] graph;

    public UCS(int nodes) {
        graph = new double[nodes][nodes];
    }

    public void addEdge(int source, int destination, double cost) {
        graph[source][destination] = cost;
    }

    public List<Integer> ucs(int source, int destination) {
        PriorityQueue<Node> pq = new PriorityQueue<>();
        boolean[] visited = new boolean[graph.length];
        int[] prev = new int[graph.length];
        double[] cost = new double[graph.length];
        Arrays.fill(cost, Double.MAX_VALUE);
        pq.add(new Node(source, 0));
        cost[source] = 0;
        while (!pq.isEmpty()) {
            Node node = pq.poll();
            int id = node.getId();
            if (visited[id]) {
                continue;
            }
            visited[id] = true;
            if (id == destination) {
                break;
            }
            for (int i = 0; i < graph.length; i++) {
                if (graph[id][i] != 0) {
                    int neighborId = i;
                    if (cost[neighborId] > cost[id] + graph[id][i]) {
                        cost[neighborId] = cost[id] + graph[id][i];
                        prev[neighborId] = id;
                        pq.add(new Node(neighborId, cost[neighborId]));
                    }
                }
            }
        }
        List<Integer> path = new ArrayList<>();
        int currentId = destination;
        while (currentId != source) {
            path.add(currentId);
            currentId = prev[currentId];
        }
        path.add(source);
        Collections.reverse(path);
        return path;
    }
}
```

```

        double neighborCost = graph[id][i];
        double newCost = cost[id] + neighborCost;
        if (newCost < cost[neighborId]) {
            cost[neighborId] = newCost;
            prev[neighborId] = id;
            pq.add(new Node(neighborId, newCost));
        }
    }
}

if (cost[destination] == Double.MAX_VALUE) {
    JPanel panel = new JPanel();
    panel.add(new JLabel("Tidak ditemukan path!"));
    JOptionPane.showMessageDialog(null, panel, "Cost
(Algoritma UCS)", JOptionPane.PLAIN_MESSAGE);
    return null;
}
JPanel panel = new JPanel();
panel.add(new JLabel("Cost : " + cost[destination]));
JOptionPane.showMessageDialog(null, panel, "Cost",
JOptionPane.PLAIN_MESSAGE);

List<Integer> path = new ArrayList<>();
int id = destination;
while (id != source) {
    path.add(id);
    id = prev[id];
}
path.add(source);
Collections.reverse(path);
return path;
}

private static class Node implements Comparable<Node> {
    private final int id;
    private final double cost;

    public Node(int id, double cost) {
        this.id = id;
        this.cost = cost;
    }
}

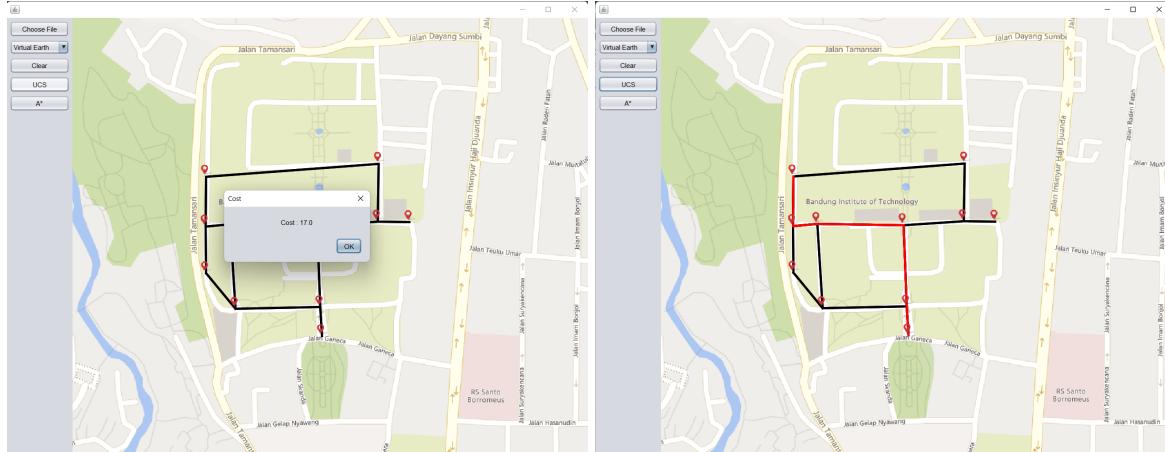
```

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

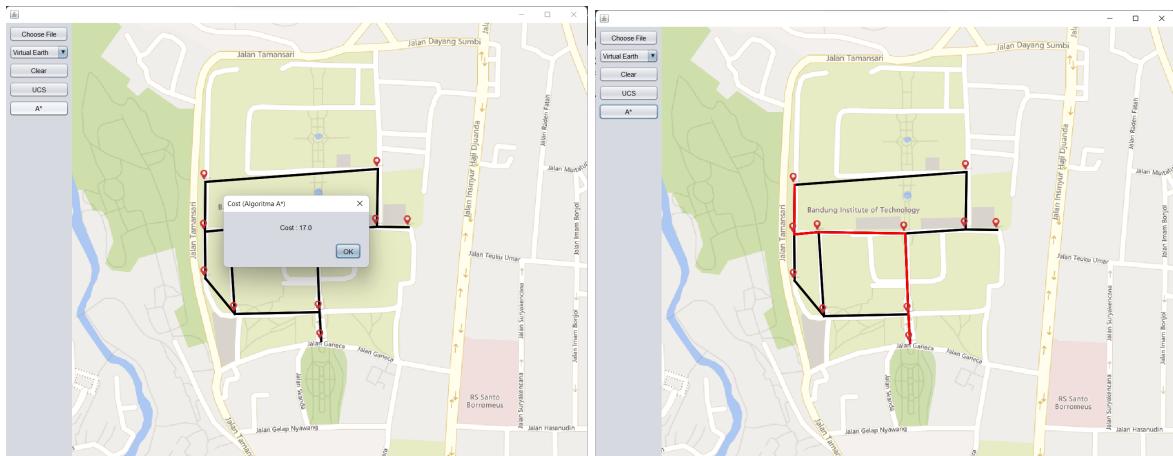
@Override
public int compareTo(Node o) {
    return Double.compare(this.cost, o.cost);
}
}
```

3 Eksperimen

3.1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara

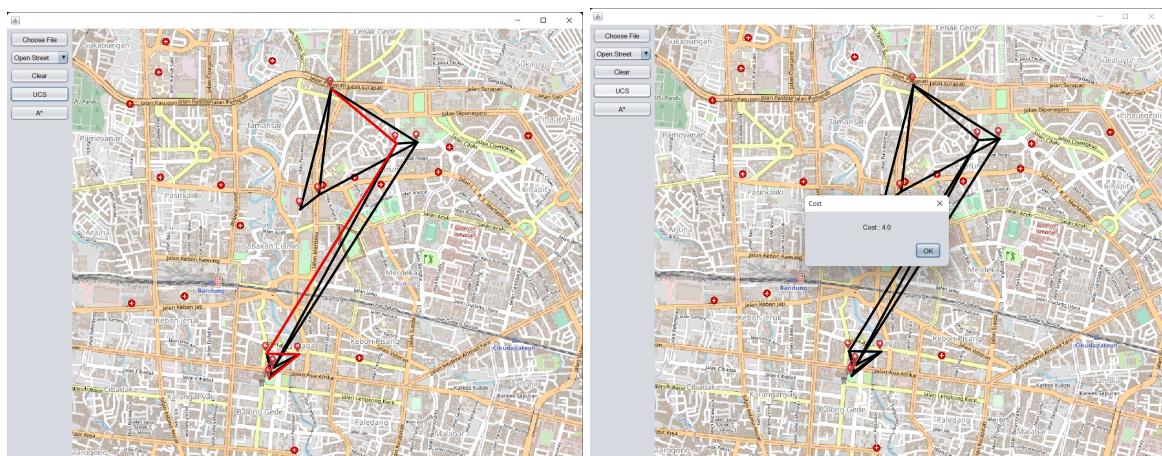


Gambar 3.1.1 UCS Cost start ke end

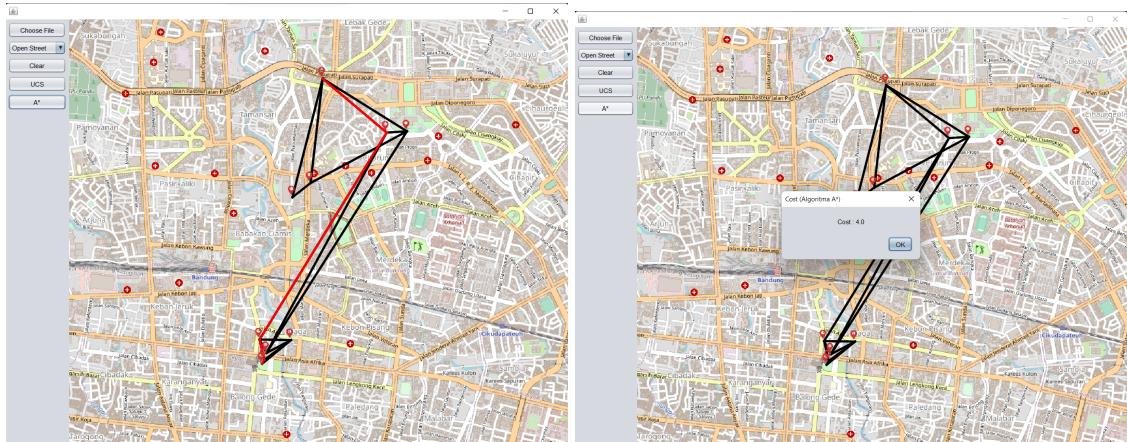


Gambar 3.1.2 A* Cost start ke end

3.2. Peta jalan sekitar Alun-alun Bandung

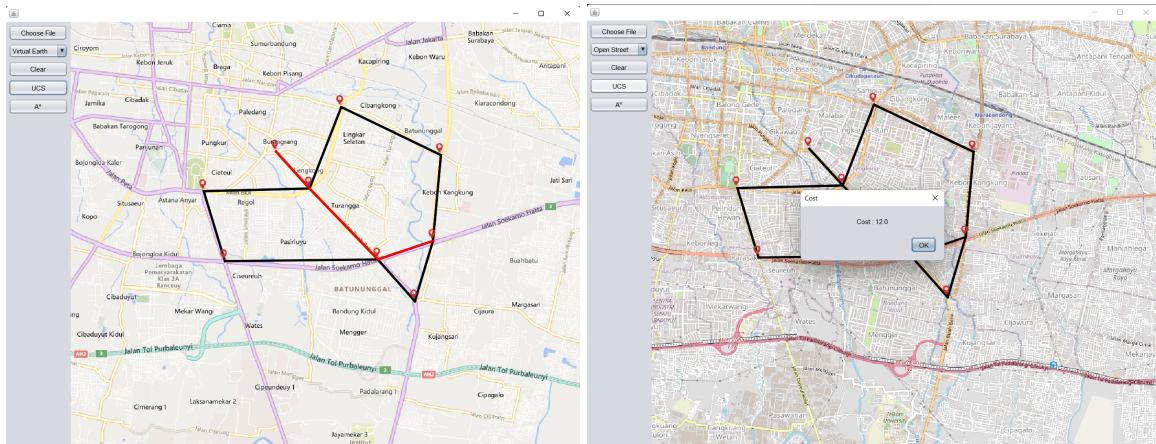


Gambar 3.2.1 UCS Cost start ke end

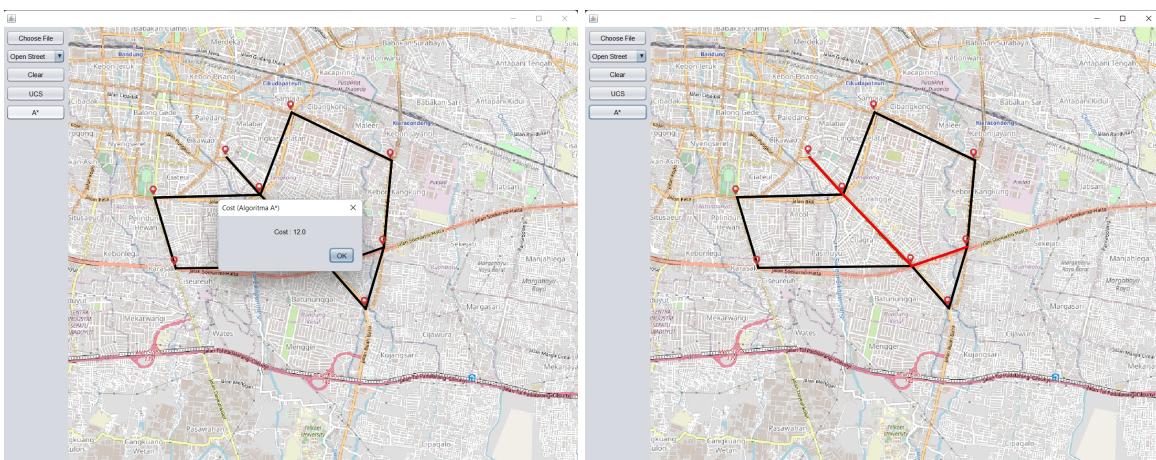


Gambar 3.2.2 A* Cost start ke end

3.3. Peta jalan sekitar Buahbatu atau Bandung Selatan

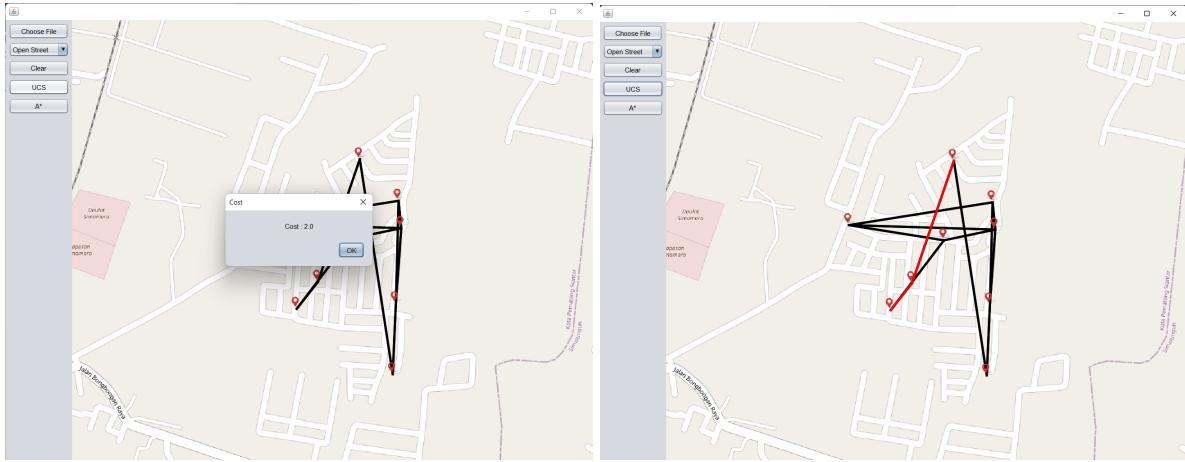


Gambar 3.3.1 UCS Cost start ke end

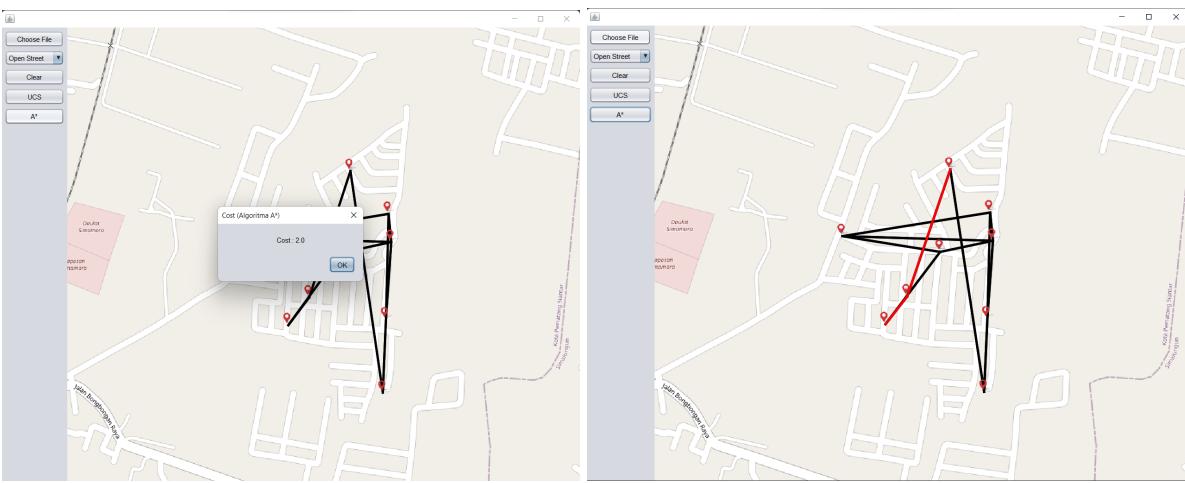


Gambar 3.3.2 A* Cost start ke end

3.4. Peta jalan sebuah kawasan di kota asalmu (Pematangsiantar)



Gambar 3.4.1 UCS Cost start ke end



Gambar 3.4.2 A* Cost start ke end

4 Repository Github

https://github.com/AlphaThrone/Tucil3_13521012_13521028

5. Kesimpulan, Komentar, dll

5.1. Kesimpulan

Dari penggerjaan tugas ini, dapat disimpulkan bahwa algoritma UCS serta algortima A* mampu mendapatkan jarak terpendek pada pencarian rute pada peta, terutama A* yang jika nilai dari setiap jarak antar simpul serta jarak tiap simpul ke titik tujuan memiliki nilai yang asli maka hasil rute yang dicari pasti optimal.

5.2. Komentar

13521012: Java “menyenangkan”

13521028: Kalau ada yang susah kenapa yang gampang.

6 Lampiran

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	✓