IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Diajukan sebagai pemenuhan tugas kecil 3.

Disusun Oleh :

1. 13521117 - Maggie Zeta RS
2. 13521164 - Akhmad Setiawan

Dosen Pengampu : Dr. Ir. Rinaldi Munir, MT.
IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

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 antar 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.

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.

BAB II
KODE PROGRAM

2.1 astar.py

File ini berisi implementasi algoritma A* yang dibuat dalam suatu implementasi kelas Astar.

```
from math import *
from classes import PriorityQueue

# find the distance on actual earth (distance on sphere)
# earth radius is approximately 6371 km
def haversine(node1, node2, radius = 6371):
    lat1 = radians(node1.latitude)
    lat2 = radians(node2.latitude)
    dLat = radians(lat2 - lat1)
    dLon = radians(node2.longitude - node1.longitude)
    a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
    c = 2*asin(sqrt(a))
    distance = radius * c
    return distance

# A* search algorithm
class Astar:
    def __init__(self, graph, start, goal):
        self.graph = graph
        self.start = start
        self.goal = goal
        self.explored = {self.start: None} # dictionary for explored
node
        self.totalCost = {self.start: 0} # dictionary for the total cost
of explored node
        self.toVisit = PriorityQueue()

    def solve(self):
        # heuristic cost from start to goal node
        self.start.cost = 0 + haversine(self.start, self.goal)
        self.toVisit.enqueue(self.start)
        while not self.toVisit.isEmpty(): # search until empty or reach
the goal node
            current = self.toVisit.dequeue()
```

```
            if current == self.goal:
                break
            for neighbor in current.neighbors:
                newCost = self.totalCost[current] + haversine(current,
neighbor)
                # update the node with lowest cost
                if (neighbor not in self.totalCost) or (newCost <
self.totalCost[neighbor]):
                    self.totalCost[neighbor] = newCost
                    neighbor.cost = newCost + haversine(neighbor,
self.goal)
                    self.toVisit.enqueue(neighbor)
                    self.explored[neighbor] = current
        # if goal node is not found, return empty
        if (self.goal not in self.explored):
            self.explored = {}
        # return node and total cost
        return self.explored, self.totalCost

    def getPath(self):
        current = self.goal
        path = [current]
        while current != self.start:
            current = self.explored[current]
            path.append(current)
        # get the path from start to goal
        path.reverse()
        return path
```

2.2 ucs.py

Sama seperti astar, file ini berisi implementasi algoritma UCS dalam instansiasi sebuah kelas UCS.

```
from classes import PriorityQueue
from astar import haversine

# Uniform-Cost-Search (UCS) algorithm
class UCS:
    def __init__(self, graph, start, goal):
        self.graph = graph
        self.start = start
        self.goal = goal
        self.explored = {self.start: None} # dictionary for explored
node
        self.totalCost = {self.start: 0} # dictionary for the total cost
of explored node
        self.toVisit = PriorityQueue()

    def solve(self):
        self.start.cost = 0
        self.toVisit.enqueue(self.start)
        while not self.toVisit.isEmpty(): # search until empty or reach
the goal node
            current = self.toVisit.dequeue()
            if current == self.goal:
                break
            for neighbor in current.neighbors:
                newCost = self.totalCost[current] + haversine(current,
neighbor)
                # update the node with lowest cost
                if neighbor not in self.totalCost or newCost <
self.totalCost[neighbor]:
                    self.totalCost[neighbor] = newCost
                    neighbor.cost = newCost
                    self.toVisit.enqueue(neighbor)
                    self.explored[neighbor] = current
        # if goal node is not found, return empty
        if (self.goal not in self.explored):
            self.explored = {}
```

```
        # return node and total cost
        return self.explored, self.totalCost

    def getPath(self):
        current = self.goal
        path = [current]
        while current != self.start:
            current = self.explored[current]
            path.append(current)
        # get the path from start to goal
        path.reverse()
        return path
```

2.3 classes.py

File ini berisi implementasi beberapa kelas yang diperlukan dalam penentuan rute, terdiri dari PriorityQueue dengan lower cost priority, Graph, dan Node.

```
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def isEmpty(self):
        return len(self.queue) == 0

    def enqueue(self, item):
        self.queue.append(item)

    # dequeue with the lowest cost priority
    def dequeue(self):
        minIdx = 0
        for i in range(len(self.queue)):
            if self.queue[i].cost < self.queue[minIdx].cost:
                minIdx = i
        val = self.queue.pop(minIdx)
        return val

class Graph(object):
    def __init__(self, nodeCount):
        self.nodeCount = nodeCount
        self.graphNodes = []

    def addNode(self, node):
        self.graphNodes.append(node)

    def printGraph(self):
        print("List of nodes:")
        for node in self.graphNodes:
            print(4*" ", end='')
            node.printNode()
        print()

    def addEdges(self, adjacencyMatrix):
        for i in range(self.nodeCount):
```

```
            for j in range(self.nodeCount):
                if adjacencyMatrix[i][j] != 0:
                    self.graphNodes[i].addNeighbor(self.graphNodes[j])

    def findNode(self, nodeName):
        for node in self.graphNodes:
            if node.name.lower() == nodeName:
                return node

class Node(object):
    def __init__(self, lat, long, name):
        self.latitude = lat
        self.longitude = long
        self.name = name
        self.cost = 0
        self.neighbors = []

    def getLatitude(self):
        return self.latitude

    def getLongitude(self):
        return self.longitude

    def printNode(self):
        print(self.name)

    def addNeighbor(self, node):
        self.neighbors.append(node)
```

2.4 utility.py

File ini berisi beberapa fungsi untuk membantu melakukan pencarian rute dari graf yang dibentuk.

```
from classes import Graph, Node
from astar import Astar
from ucs import UCS
import sys

# graph initializer from file.txt
def initializeGraph(filename):
    text = filename.read().splitlines()
    nodeCount = int(text[0])
    if (nodeCount < 8):
        print("Please fix your file, the nodes must be greater or equal
than 8")
        sys.exit()
    nodes = text[1:nodeCount+1]
    fileMatrix = text[nodeCount+1:]
    adjacencyMatrix = [[0 for i in range(nodeCount)] for j in
range(nodeCount)]

    # input searching method
    while True:
        method = input("Input Method (ucs/astar): ")
        if method.lower() == "ucs":
            method = "ucs"
            break
        elif method.lower() == "astar":
            method = "astar"
            break
        else:
            print("Invalid Method!")

    graph = Graph(nodeCount)
    # add node from file.txt
    for i in range(nodeCount):
        splitString = nodes[i].split(" ", 2)
        newNode = Node(float(splitString[0]), float(splitString[1]),
splitString[2])
```

```
        graph.addNode(newNode)
        adjacencyMatrix[i] = [int(x) for x in fileMatrix[i].split(" ")]
    graph.addEdges(adjacencyMatrix)
    return graph, method

# lat and long getter
def solveGraph(graph):
    latitude = []
    longitude = []
    for i in range(len(graph.graphNodes)):
        latitude.append(graph.graphNodes[i].latitude)
        longitude.append(graph.graphNodes[i].longitude)
    return latitude, longitude

# neighbor solver
def solveNeighbor(graph):
    arrDict = []
    nodes = graph.graphNodes
    for i in range(len(nodes)):
        arrDict.append({
            'latitude': nodes[i].latitude,
            'longitude': nodes[i].longitude,
            'neighbor': []
        })
        for j in range(len(nodes[i].neighbors)):
            arrDict[i]['neighbor'].append({
                'latitude': nodes[i].neighbors[j].latitude,
                'longitude': nodes[i].neighbors[j].longitude
            })
    return arrDict

# path solver
def solvePath(path):
    name = []
    latitude = []
    longitude = []
    for i in range(len(path)):
        name.append(path[i].name)
        latitude.append(path[i].latitude)
```

```
        longitude.append(path[i].longitude)
    # get names
    node = dict(enumerate(name))
    names = []
    names.append(node)
    # get latitudes
    lat = dict(enumerate(latitude))
    latitudes = []
    latitudes.append(lat)
    # get longitudes
    long = dict(enumerate(longitude))
    longitudes = []
    longitudes.append(long)
    return names, latitudes, longitudes

def main(graph, method):
    print("Input location")
    start = None
    goal = None
    while start == None or goal == None:
        # get start node
        while start == None:
            startNode = input("    Start Location: ")
            start = graph.findNode(startNode.lower())
            if (start == None):
                print("Invalid Location! Try Again")
                print()
        # get goal node
        while goal == None:
            goalNode = input("    Goal Location: ")
            goal = graph.findNode(goalNode.lower())
            if (goal == None):
                print("Invalid Location! Try Again")
                print()

    # using ucs alogrithm
    if method == "ucs":
        ucs = UCS(graph, start, goal)
        explored, totalCost = ucs.solve()
```

```
        if (len(explored) == 0):
            path = []
            distance = 0
        else:
            path = ucs.getPath()
            distance = totalCost[goal]

    # using astar algorithm
    else:
        astar = Astar(graph, start, goal)
        explored, totalCost = astar.solve()
        if (len(explored) == 0):
            path = []
            distance = 0
        else:
            path = astar.getPath()
            distance = totalCost[goal]
    return start, goal, path, distance

def printSolution(graph, method):
    start, goal, path, distance = main(graph, method)
    print("\n====[ SHORTEST PATH FROM " + start.name.upper() + " TO " +
goal.name.upper() + " ]====")
    print("METHOD: " + method.upper())
    print("ROUTE: ", end="")
    for node in path:
        if node.name != goal.name:
            print(node.name, end=" - ")
        else:
            print(node.name)
    print("TOTAL DISTANCE: " + str(round(distance*1000, 3)) + " m")
```

2.5 main.py

Merupakan file program utama yang akan dijalankan dengan Flask.

```
import json
import sys
from utility import *
from classes import *
from astar import Astar
from ucs import UCS
from flask import Flask, render_template
from flask.json import jsonify

app = Flask(__name__)
# file input validation
while True:
    filename = input("Input filename: ")
    try:
        file = open("test/" + filename)
        break
    except FileNotFoundError:
        print("File Not Found!\n")

graph, method = initializeGraph(file)
graph.printGraph()

while True:
    visualize = input("Visualize the solution with google maps API?
(y/n): ")
    if visualize.lower() == "n" or visualize.lower() == "no":
         printSolution(graph, method)
         sys.exit()
    elif visualize.lower() == "y" or visualize.lower() == "yes":
        print("Run http://127.0.0.1:5000 in your local browser, then
enter start and goal node")
        print()
        break
    else:
        print("Invalid Input!\n")

# Flask
```

```
@app.route("/get-data")
def getData():
    arrDict = solveNeighbor(graph)
    return jsonify(arrDict = arrDict)

@app.route("/")
def init():
    start, goal, path, distance = main(graph, method)
    dict, lat, long = solvePath(path)
    markLatitude, markLongitude = solveGraph(graph)
    solution = json.dumps(dict)
    latitudes = json.dumps(lat)
    longitudes = json.dumps(long)
    startLatitude = start.latitude
    startLongitude = start.longitude
    goalLatitude = goal.latitude
    goalLongitude = goal.longitude
    paths = path
    return render_template(
        'visualization.html',
        method = method.upper(),
        startLat = startLatitude,
        startLong = startLongitude,
        goalLat = goalLatitude,
        goalLong = goalLongitude,
        solution = solution,
        solutionPath = paths,
        latitudes = latitudes,
        longitudes = longitudes,
        markLatitude = markLatitude,
        markLongitude = markLongitude,
        dist = round(distance*1000, 3))
```

2.6 visualization.html

Merupakan file yang berisi kode html dalam menampilkan Google Maps dan Solusi Rutenya

```
<!DOCTYPE html>
<html>
    <head>
        <meta name="viewport" content="width=device-width,
initial-scale=1">
        <meta charset="UTF-8">
        <title>UCS and A* Searching Algorithm</title>
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.m
in.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9Jv
oRxT2MZw1T" crossorigin="anonymous">
        <script async defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyC8Ti9b-rHdaKt81gw
PugA2Gl291hB_qp4&libraries=geometry&callback=initMap"
        type="text/javascript"></script>
        <script type="text/javascript">
            window.initMap = function() {
                var labelIndex = 0;
                var directionsService = new
google.maps.DirectionsService;
                var directionsDisplay = new
google.maps.DirectionsRenderer;
                const nodeCode = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                var map = new
google.maps.Map(document.getElementById('map'), {
                    center: {
                        lat: {{startLat}},
                        lng: {{startLong}}
                    },
                    zoom: 16
                });
                directionsDisplay.setMap(map);
                // add label for solution
                function addLabel(coordinate){
                    var marker = new google.maps.Marker({
                        position: coordinate,
```

```
                        label: { text: nodeCode[labelIndex++ %
nodeCode.length],
                                color: 'black',
                                fontWeight: 'bold'
                                },
                        map: map,
                    });
                }
                // add marker for nodes
                function addNode(coordinate){
                    var marker = new google.maps.Marker({
                        position: coordinate,
                        map: map,
                    });
                }
                var markLatitude = {{markLatitude}};
                var markLongitude = {{markLongitude}};
                for(var i = 0; i < markLatitude.length; i++){
                    addNode({lat: markLatitude[i], lng:
markLongitude[i]});
                }
                var points = JSON.parse('{{solution | safe}}');
                var lats = JSON.parse('{{latitudes | safe}}');
                var longs = JSON.parse('{{longitudes | safe}}');
                for(var i = 0; i < Object.keys(lats[0]).length; i++){
                    addLabel({lat: lats[0][i], lng: longs[0][i]});
                }
                var graphObj;
                fetch('get-data')
                .then(response => response.json())
                .then(data => graphObj = data)
                .then(() => {
                    const nodeList = graphObj.arrDict;
                    for (let i = 0; i < nodeList.length; i++) {
                        for (let j = 0; j < nodeList[i].neighbor.length;
j++) {
                            let arrayPath = [];
                            arrayPath.push({
                                lat: parseFloat(nodeList[i].latitude),
```

```
                                lng: parseFloat(nodeList[i].longitude)
                            }, {
                                lat:
parseFloat(nodeList[i].neighbor[j].latitude),
                                lng:
parseFloat(nodeList[i].neighbor[j].longitude)
                            })
                            // draw edges
                            var drawEdge = new google.maps.Polyline({
                                path: arrayPath,
                                geodesic: true,
                                strokeColor: "#0000FF",
                                strokeOpacity: 1.0,
                                strokeWeight: 2,
                            });
                            // draw solution
                            var drawLine = new google.maps.Polyline({
                                path: coordinateArray,
                                strokeColor: "#FF0000",
                                strokeOpacity: 1.0,
                                strokeWeight: 6,
                            });
                            drawEdge.setMap(map);
                            drawLine.setMap(map);
                        }
                    }
                })
                // fill array as latitude and longitude from solution
node
                var coordinateArray = [];
                for(var i = 0; i < Object.keys(lats[0]).length; i++){
                    coordinateArray.push({
                        lat: lats[0][i],
                        lng: longs[0][i]
                    });
                }
            }
        </script>
    </head>
```

```
    <body class="d-flex h-100 text-center text-black bg-light">
        <div class="container">
            <header class="pt-4 pb-2 mb-4 bg-light text-black">
                <div>
                    <h1>UCS AND A* FINDING ALGORITHM</h1>
                </div>
            </header>
            <div class="mb-5 bg-light rounded-3">
                <div id="map" style="width:100%; height:500px;"></div>
            </div>
            <div class="row">
                <div class="col-lg-6">
                    <h2 class="pt-3">ROUTE PATH:</h2>
                        {% for path in solutionPath %}
                        <h3>{{path.name}}</h3>
                        {% endfor %}
                </div>
                <div class="col-lg-6">
                    <h2 class="pt-3">DISTANCE:</h2>
                    <h3>{{dist}} m</h3>
                </div>
            </div>
        </div>
    </body>
</html>
```

BAB III
IMPLEMENTASI UJI KASUS

1. Peta Jalan Sekitar Kampus ITB

File test/itb.txt:

```
8
-6.884589 107.609801 Jl. Siliwangi
-6.887741 107.609578 Jl. Tamansari
-6.887292 107.611562 Jl. Dayang Sumbi
-6.893853 107.608408 Kebun Binatang
-6.893257 107.610468 Jl. Ganesha
-6.896877 107.609589 Pelesiran
-6.898786 107.607388 Jl. Pasopati
-6.894803 107.610286 Gelap Nyawang
0 0 1 0 0 0 0 0
0 0 1 1 0 0 0 0
1 1 0 0 0 0 0 0
0 1 0 0 1 1 0 1
0 0 0 1 0 0 0 1
0 0 0 1 0 0 1 1
0 0 0 0 0 1 0 0
0 0 0 1 1 1 0 0
```

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: itb.txt
Input Method (ucs/astar): ucs
List of nodes:
    Jl. Siliwangi
    Jl. Tamansari
    Jl. Dayang Sumbi
    Kebun Binatang
    Jl. Ganesha
    Pelesiran
    Jl. Pasopati
    Gelap Nyawang

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: Jl. Pasopati
    Goal Location: Jl. Dayang Sumbi

====[ SHORTEST PATH FROM JL. PASOPATI TO JL. DAYANG SUMBI ]====
METHOD: UCS
ROUTE: Jl. Pasopati - Pelesiran - Kebun Binatang - Jl. Tamansari - Jl. Dayang Sumbi
TOTAL DISTANCE: 722.222 m
```

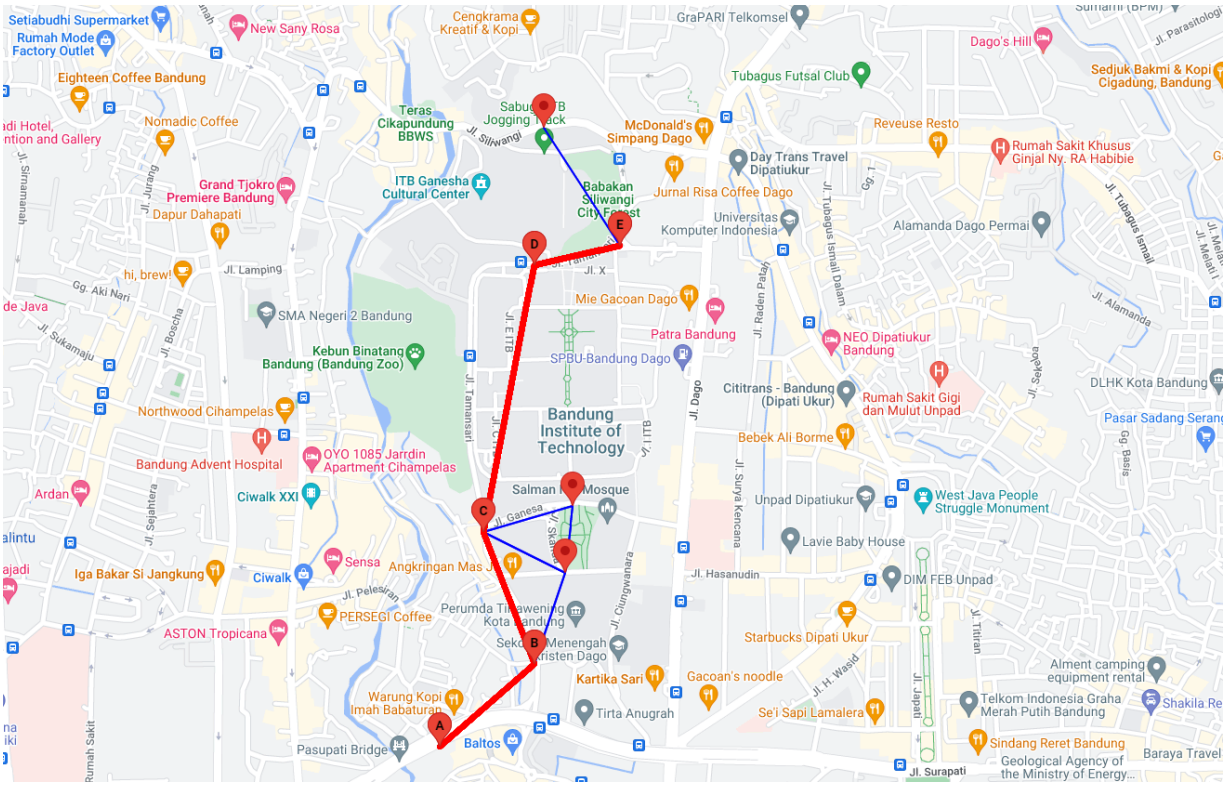
Metode UCS

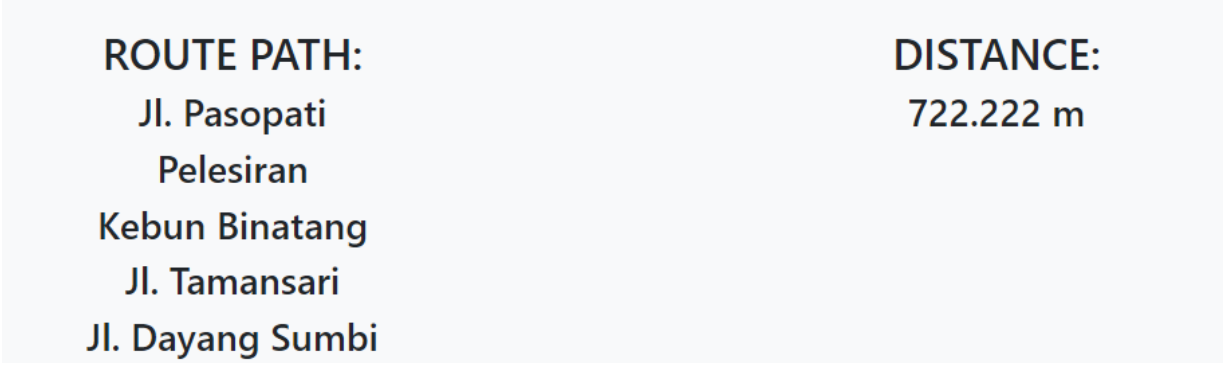
```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: itb.txt
Input Method (ucs/astar): astar
List of nodes:
    Jl. Siliwangi
    Jl. Tamansari
    Jl. Dayang Sumbi
    Kebun Binatang
    Jl. Ganesha
    Pelesiran
    Jl. Pasopati
    Gelap Nyawang

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: Jl. Pasopati
    Goal Location: Jl. Dayang Sumbi

====[ SHORTEST PATH FROM JL. PASOPATI TO JL. DAYANG SUMBI ]====
METHOD: ASTAR
ROUTE: Jl. Pasopati - Pelesiran - Kebun Binatang - Jl. Tamansari - Jl. Dayang Sumbi
TOTAL DISTANCE: 722.222 m
```

Metode A*





Visualisasi dengan GMaps

2. **Peta Jalan Sekitar Alun-alun Bandung**

File test/alunAlun.txt:

```
10
-6.920818 107.604101 Jend Sudirman Navaro
-6.920892 107.605089 Kontiki Lintas Media
-6.921052 107.606473 Toko Komputer New Pelangi
-6.921229 107.607691 Masjid Agung
-6.922082 107.604025 Garuda Kencana Toko
-6.922383 107.606428 Dalem Kaum
-6.922522 107.607099 Pendopo Bandung
-6.922552 107.607521 Dalem Kaum
-6.923108 107.603929 Otista Kepatihan
-6.923422 107.606289 Balonggede
0 1 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0 1 0
0 0 0 0 1 0 1 0 0 1
0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 1 0 0 1 0
```

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: alunALun.txt
Input Method (ucs/astar): ucs
List of nodes:
    Jend Sudirman Navaro
    Kontiki Lintas Media
    Toko Komputer New Pelangi
    Masjid Agung
    Garuda Kencana Toko
    Dalem Kaum I
    Pendopo Bandung
    Dalem Kaum II
    Otista Kepatihan
    Balonggede

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: Masjid agung
    Goal Location: balonggede

====[ SHORTEST PATH FROM MASJID AGUNG TO BALONGGEDE ]====
METHOD: UCS
ROUTE: Masjid Agung - Dalem Kaum II - Pendopo Bandung - Dalem Kaum I - Balonggede
TOTAL DISTANCE: 155.066 m
```

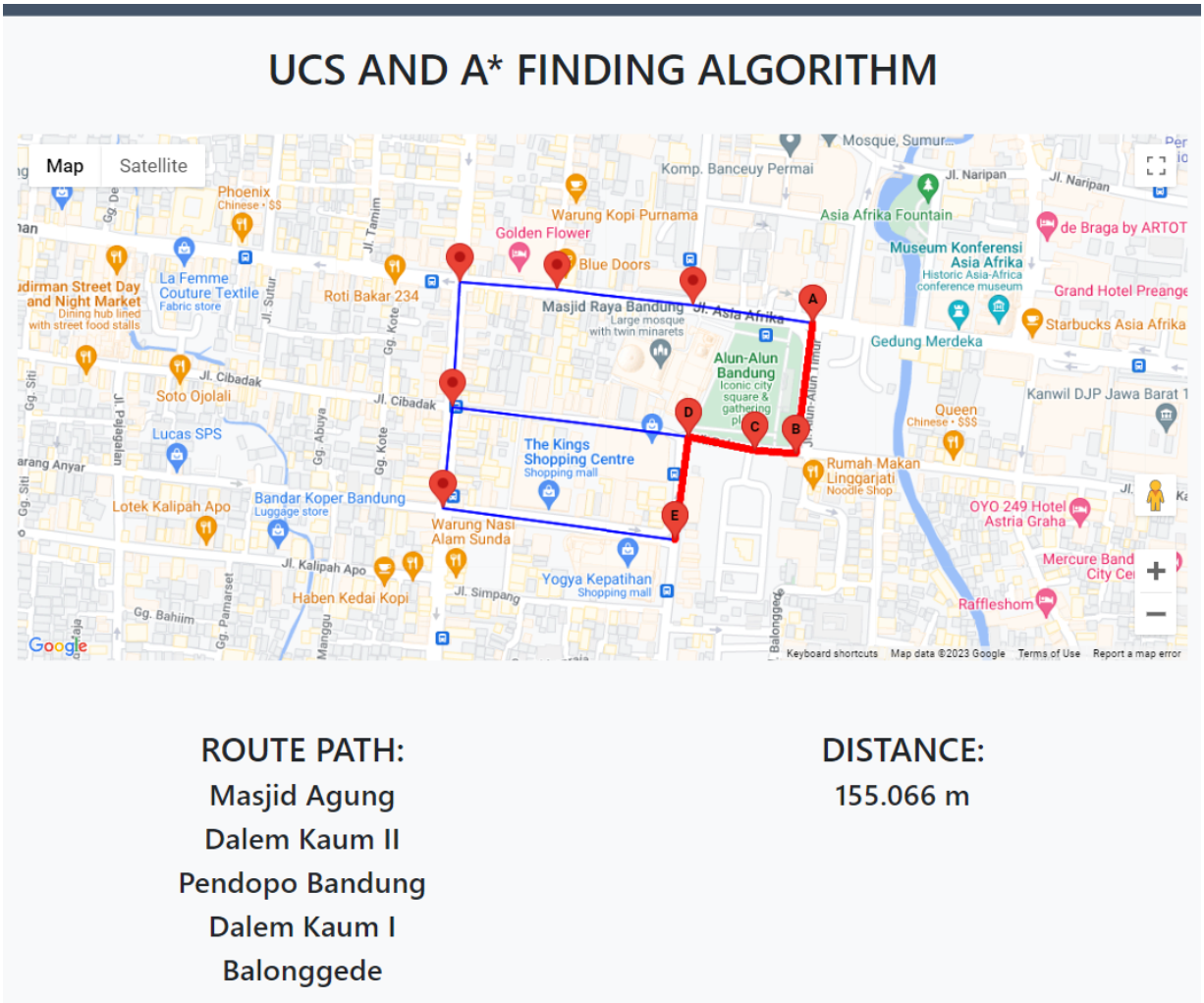
Metode UCS

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: alunAlun.txt
Input Method (ucs/astar): astar
List of nodes:
    Jend Sudirman Navaro
    Kontiki Lintas Media
    Toko Komputer New Pelangi
    Masjid Agung
    Garuda Kencana Toko
    Dalem Kaum I
    Pendopo Bandung
    Dalem Kaum II
    Otista Kepatihan
    Balonggede

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: masjid agung
    Goal Location: balonggede

====[ SHORTEST PATH FROM MASJID AGUNG TO BALONGGEDE ]====
METHOD: ASTAR
ROUTE: Masjid Agung - Dalem Kaum II - Pendopo Bandung - Dalem Kaum I - Balonggede
TOTAL DISTANCE: 155.066 m
```

Metode A*



Visualisasi dengan GMaps

3. Peta Jalan Sekitar Buah Batu

File test/buahBatu.txt

```
8
-6.931832 107.618105 Jl. Banteng
-6.932927 107.618847 Jl. Buah Batu
-6.929883 107.619782 Jl. Lodaya
-6.928028 107.619618 Jl. Talaga Bodas
-6.931533 107.616036 Jl. Gurame
-6.929681 107.616165 Jl. Karapitan
-6.928093 107.618954 Jl. Sadakeling
-6.933481 107.622872 Jl. KH. Ahmad Dahlan
0 1 1 0 0 0 0 1
1 0 0 0 1 0 1 1
```

```
1 0 0 1 0 0 1 1
0 0 1 0 0 0 1 0
0 1 0 0 0 1 1 0
0 0 0 0 1 0 0 0
0 1 1 1 1 0 0 0
1 1 1 0 0 0 0 0
```

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: buahBatu.txt
Input Method (ucs/astar): ucs
List of nodes:
    Jl. Banteng
    Jl. Buah Batu
    Jl. Lodaya
    Jl. Talaga Bodas
    Jl. Gurame
    Jl. Karapitan
    Jl. Sadakeling
    Jl. KH. Ahmad Dahlan

Visualize the solution with google maps API? (y/n): Jl. banteng
Invalid Input!

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: Jl. banteng
    Goal Location: jl. karapitan

====[ SHORTEST PATH FROM JL. BANTENG TO JL. KARAPITAN ]====
METHOD: UCS
ROUTE: Jl. Banteng - Jl. Buah Batu - Jl. Gurame - Jl. Karapitan
TOTAL DISTANCE: 406.913 m
```

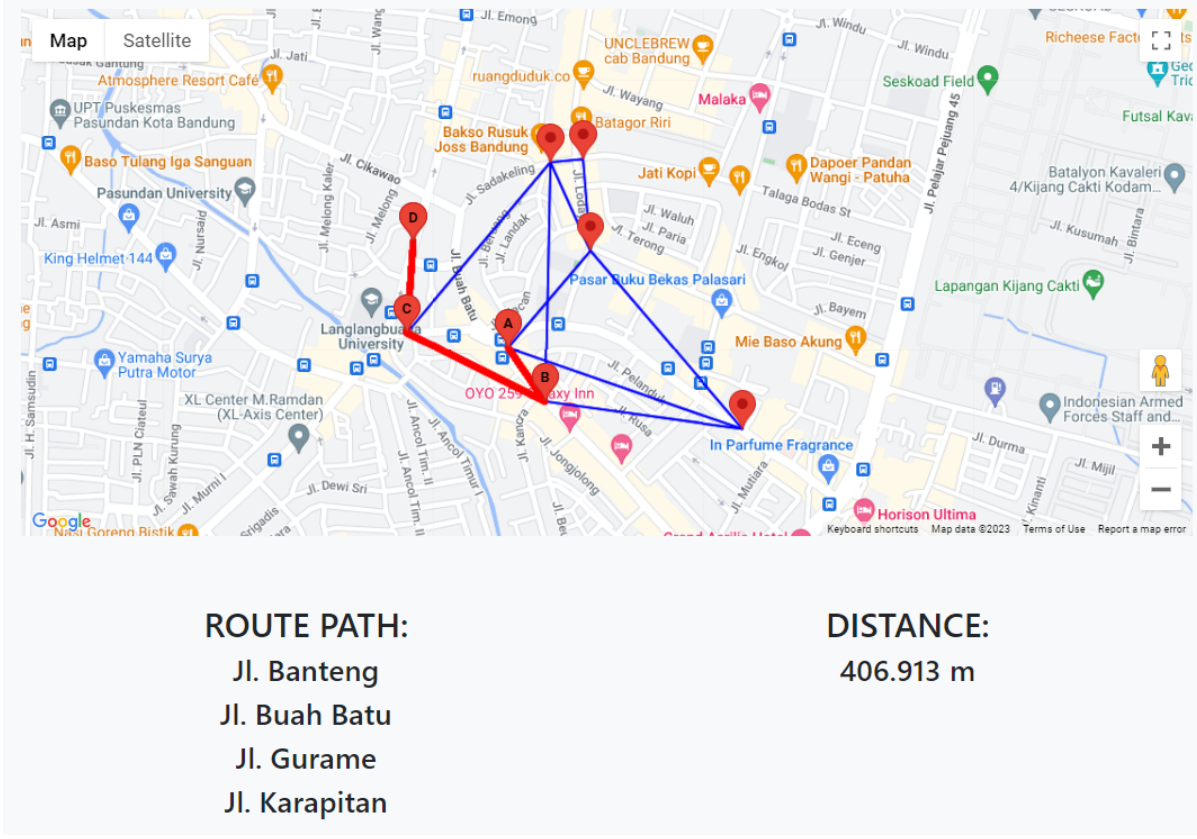
Metode UCS

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: buahBatu.txt
Input Method (ucs/astar): astar
List of nodes:
    Jl. Banteng
    Jl. Buah Batu
    Jl. Lodaya
    Jl. Talaga Bodas
    Jl. Gurame
    Jl. Karapitan
    Jl. Sadakeling
    Jl. KH. Ahmad Dahlan

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: jl. banteng
    Goal Location: jl. karapitan

====[ SHORTEST PATH FROM JL. BANTENG TO JL. KARAPITAN ]====
METHOD: ASTAR
ROUTE: Jl. Banteng - Jl. Buah Batu - Jl. Gurame - Jl. Karapitan
TOTAL DISTANCE: 406.913 m
```

Method A*



Visualisasi dengan GMaps

4. **Peta Jalan Sekitar Simpang Lima Semarang**

File test/semarang.txt

```
9
-6.990486 110.423838 Jl. Ahmad Yani
-6.997268 110.419266 Jl. Pahlawan
-6.993383 110.421671 Jl. Imam Barjo
-6.995822 110.414089 Jl. Veteran
-6.993415 110.413746 Jl. Dokter Kariadi
-6.993266 110.419242 Jl. Mentri Supeno
-6.991624 110.420101 Jl. Pandanaran II
-6.988793 110.420098 Jl. Pandanaran
-6.989984 110.422202 Simpang Lima
0 0 1 0 0 0 0 0 1
0 0 1 1 0 0 0 0 0
1 1 0 0 0 1 0 0 1
```

```
0 1 0 0 1 0 0 0 0
0 0 0 1 0 1 0 0 0
0 0 1 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 1 0 1
1 0 1 0 0 0 0 1 0
```

```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: semarang.txt
Input Method (ucs/astar): ucs
List of nodes:
    Jl. Ahmad Yani
    Jl. Pahlawan
    Jl. Imam Barjo
    Jl. Veteran
    Jl. Dokter Kariadi
    Jl. Mentri Supeno
    Jl. Pandanaran II
    Jl. Pandanaran
    Simpang Lima

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: Jl. ahmad yani
    Goal Location: jl. veteran

====[ SHORTEST PATH FROM JL. AHMAD YANI TO JL. VETERAN ]====
METHOD: UCS
ROUTE: Jl. Ahmad Yani - Jl. Imam Barjo - Jl. Pahlawan - Jl. Veteran
TOTAL DISTANCE: 1076.15 m
```

Method UCS

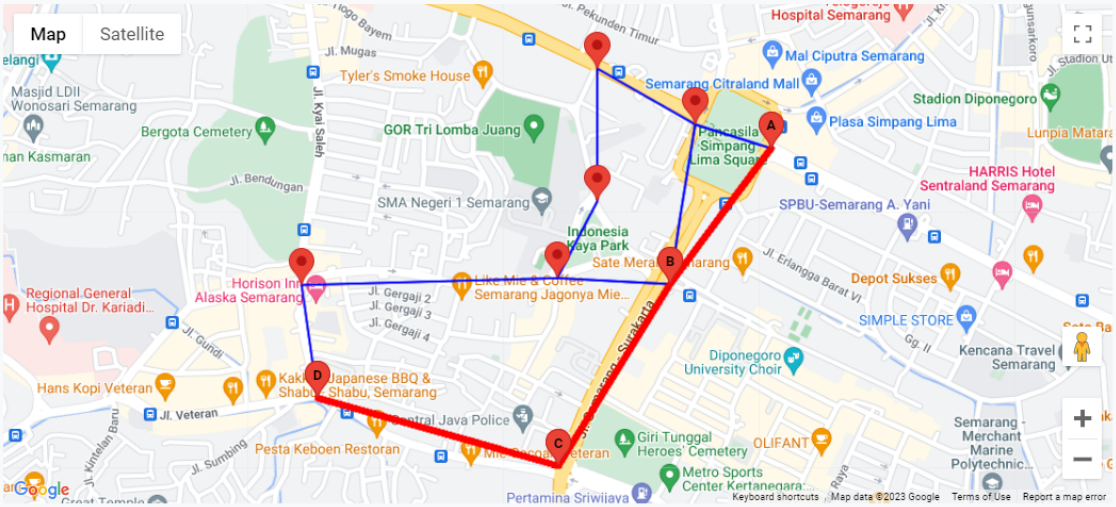
```
(env) C:\Users\Lenovo\Documents\Semester 4\Stima\Tucil\Tucil3_Stima>flask run
Input filename: semarang.txt
Input Method (ucs/astar): astar
List of nodes:
    Jl. Ahmad Yani
    Jl. Pahlawan
    Jl. Imam Barjo
    Jl. Veteran
    Jl. Dokter Kariadi
    Jl. Mentri Supeno
    Jl. Pandanaran II
    Jl. Pandanaran
    Simpang Lima

Visualize the solution with google maps API? (y/n): n
Input location
    Start Location: jl. ahmad Yani
    Goal Location: jl. veteran

====[ SHORTEST PATH FROM JL. AHMAD YANI TO JL. VETERAN ]====
METHOD: ASTAR
ROUTE: Jl. Ahmad Yani - Jl. Imam Barjo - Jl. Pahlawan - Jl. Veteran
TOTAL DISTANCE: 1076.15 m
```

Method A*

UCS AND A* FINDING ALGORITHM



ROUTE PATH:

Jl. Ahmad Yani
Jl. Imam Barjo
Jl. Pahlawan
Jl. Veteran

DISTANCE:

1076.15 m

Visualisasi dengan GMaps

BAB IV
KESIMPULAN, SARAN, DAN REFLEKSI

Dari Tugas Kecil 3 IF2211 Strategi Algoritma Semester II 2022/2023 berjudul *Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek*, kami mendapati bahwa untuk menentukan lintasan terpendek dapat dilakukan dengan menggunakan algoritma UCS dan A*. Berdasarkan hasil implementasi, dapat ditarik kesimpulan bahwa menggunakan metode pencarian jalur terpendek dengan algoritma A* lebih efektif dibandingkan dengan menggunakan metode pencarian jalur terpendek dengan algoritma UCS. Hal ini terlihat dari hasil pengujian pada salah satu test case di atas, di mana jarak yang ditempuh dengan menggunakan metode A* lebih pendek daripada menggunakan metode UCS. Meskipun ada beberapa test case yang menunjukkan bahwa jarak yang ditempuh sama antara kedua metode, namun penggunaan metode A* mengunjungi jumlah node yang lebih sedikit dibandingkan dengan metode UCS, menunjukkan bahwa waktu eksekusi dari metode A* lebih cepat daripada metode UCS.

Saran-saran yang dapat kami berikan untuk Tugas Kecil 3 IF2211 Strategi Algoritma Semester II 2022/2023 adalah algoritma UCS dan A* yang digunakan dalam tugas kecil ini masih dapat dikembangkan untuk kepentingan efisiensi program karena tentu masih terdapat kekurangan. Selain itu, untuk memudahkan programmer me-maintenance program yang ada, ada baiknya kode program dibuat lebih modular dan tersegmentasi dengan baik. Terakhir, program ini dapat dipublikasikan setelah dikembangkan lebih lanjut agar dapat bermanfaat menjadi referensi publik.

Setelah menyelesaikan Tugas Kecil 3 IF2211 Strategi Algoritma Semester II 2022/2023, kami dapat merefleksikan bahwa komunikasi antaranggota kelompok berjalan cukup baik sehingga tidak terjadi miskomunikasi dan kesalahpahaman dalam pengerjaan tugas kecil ini. Sebelum dimulainya pengerjaan tugas kecil ini juga kami melakukan diskusi untuk membahas pembagian kerja untuk setiap anggota kelompok.

LAMPIRAN

Link repository : [blixa-rd/Tucil3_13521117_13521164: Tugas Kecil 3 Strategi Algoritma - Shortest Path Finding with UCS and A* Algorithm (github.com)]

1	Program dapat menerima input graf	V
2	Program dapat menghitung lintasan terpendek dengan UCS	V
3	Program dapat menghitung lintasan terpendek dengan A*	V
4	Program dapat menampilkan lintasan terpendek serta jaraknya	V
5	Bonus: Program dapat menampilkan peta serta lintasan terpendek pada peta	V