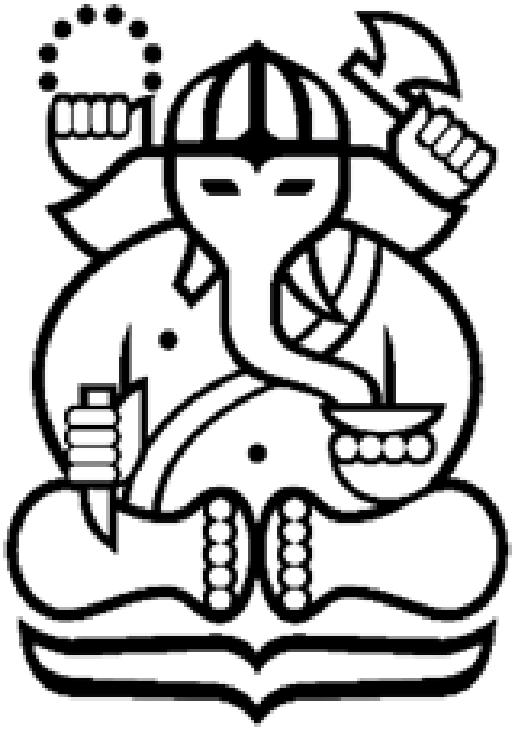
LAPORAN TUGAS KECIL III
IF2211 STRATEGI ALGORITMA

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek



Disusun oleh

Jeffrey Chow	13521046
Marcel Ryan Antony	13521127

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKOLOGI BANDUNG

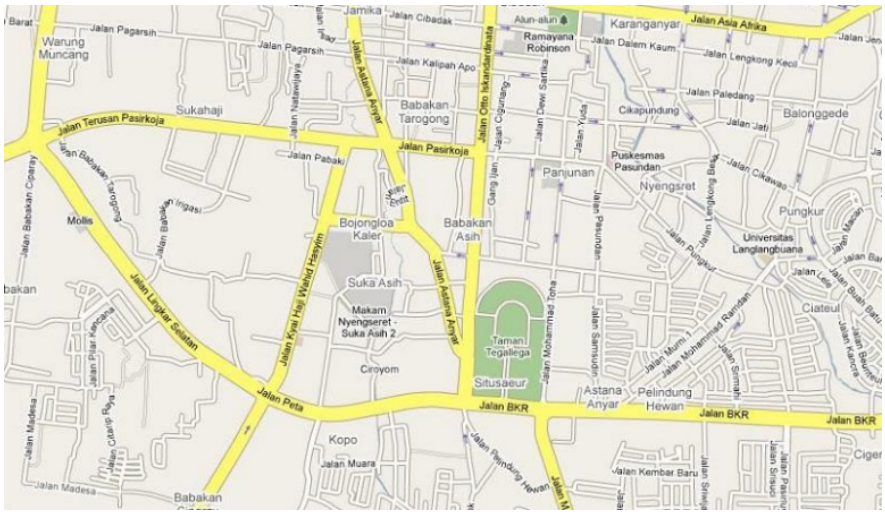
2023

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.



Gambar 1.1 Google maps

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.

Spesifikasi Program :

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf.
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

BAB II

KODE PROGRAM

Program dibuat dengan menggunakan bahasa Python dan terdiri dari 9 modul. Modul-modul tersebut adalah mainProgram.py (program utama), app.py (visualizer peta dan graf), parse.py (parsing file menjadi matriks tetangga dan graf), haversine.py (fungsi untuk mencari jarak antara dua titik pada peta), astar.py, ucs.py (file program algoritma), Graph.py , Node.py, Location.py (kelas Graph, Node, dan Location). Berikut kode-kode program diatas :

mainProgram.py

```
from datatype.Graph import Graph
from lib.astar import astar
from lib.ucs import ucs
import os

def main_program():
    # HEADER
    print('''\n
============================================
UCS and A* Algorithm for Closest Path Finding
============================================
    ''')

    # GET ALL TEST CASE THAT IS AVAILABLE
    path = r".\tests"
    testCaseList = os.listdir(path)

    # PRINT ALL AVAILABLE TEST CASES
    print("LIST OF TEST CASES : ")
    for i in range(len(testCaseList)):
        print(f"[{i+1}] {testCaseList[i]}")
    print("\n")
    # INPUT FILENAME INDEX
    filenameIndex = int(input("Input filename number (based on list above) : "))

    # VALIDATING FILENAME INDEX INPUT
    while (filenameIndex > len(testCaseList) or filenameIndex <= 0):
        print("Input number based on numbers of above list!!")
        filenameIndex = int(input("Input filename number (based on list above) : "))

    # INITIALIZE GRAPH
    filename = testCaseList[filenameIndex-1]
```

```
    graph = Graph(filename)

    # PRINT ALL LOCATION NAME AVAILABLE IN FILE
    print(f'''ALL LOCATIONS IN {filename} :''')
    graph.printNames()
    print("\n")
    # INPUT SOURCE INDEX
    sourceIndex = int(input('Input source location number : '))

    # VALIDATING SOURCE INDEX INPUT
    while (sourceIndex > len(graph.locations) or sourceIndex <= 0):
        print("Input number based on the location number above")
        sourceIndex = int(input('Input source location number : '))


    # INPUT DESTINATION INDEX
    destinationIndex = int(input('Input destination location number : '))

    # VALIDATING DESTINATION INDEX INPUT
    while (destinationIndex > len(graph.locations) or destinationIndex <= 0):
        print("Input number based on the location number above")
        destinationIndex = int(input('Input destination location number : '))

    # GET SOURCE AND DESTINATION LOCATION
    source = graph.locations[sourceIndex-1].name
    destination = graph.locations[destinationIndex-1].name

    # INPUT ALGORITHM CHOICE
    print('''\n
Algorithm :
[1] Uniform Cost Search (UCS)
[2] A*
    ''')
    choice = -1
    while choice < 1 or choice > 2 :
        choice = int(input("Input algorithm choices : "))

    # SELECT ALGORITHM
    result = None
    if choice == 1:
        result = ucs(graph, source, destination)
        distance = result[0]
        route = result[2]
    else :
        result = astar(graph, source, destination)
```

```
        distance = result.f
        route = result.route.replace(' ', ' - ')

    # PRINT RESULT
    print(f'''
============ RESULT ============
Filename        : {filename}
Source          : {source}
Destination     : {destination}
Distance (km)   : {distance}
Route           : {route}
    ''')

    print('''
TO VIEW VISUALIZER:
[1] In web browser go to localhost:5000
[2] Red Point   : Source Location
    Green Point : Destination Location
    Blue Point  : Regular Location
    Blue Line   : Shortest Route from source to destination
    Green Line  : Regular Route
[3] Hover on each Point to see the location name
    ''')

    # DATA MANIPULATION FOR VISUALIZING
    shortest_route_name_list = route.split(' - ')

    # CREATE LIST OF ROUTES BETWEEN EACH ADJACENT IN SHORTEST ROUTE NAME LIST
    shortest_routes = []
    for i in range(len(shortest_route_name_list) - 1):
        start_loc = graph.findLocation(shortest_route_name_list[i])
        end_loc = graph.findLocation(shortest_route_name_list[i+1])
        start = (start_loc.getLat(), start_loc.getLong())
        end = (end_loc.getLat(), end_loc.getLong())
        route = [start, end]
        shortest_routes.append(route)
        shortest_routes.append(list(reversed(route)))

    # GET SOURCE AND DESTINATION LOCATION
    source_loc = graph.findLocation(source)
    destination_loc = graph.findLocation(destination)

    return graph.locations, graph.adjacencyMatrix, source_loc, destination_loc,
shortest_routes
```

app.py

```
from flask import Flask, render_template
import folium
from mainProgram import main_program

app = Flask(__name__)

# GLOBAL VARIABLE RESULT TO VISUALIZE
result = None

@app.route('/')
def index():
    # FETCH DATAS
    locations, adjacency_matrix, source_loc, destination_loc, shortest_routes =
result

    # FIND ALL ROUTES BASED ON ADJACENCY MATRIX
    routes = []
    for i in range(len(adjacency_matrix)):
        for j in range(i+1, len(adjacency_matrix[i])):
            if adjacency_matrix[i][j] > 0:
                start = (locations[i].getLat(), locations[i].getLong())
                end = (locations[j].getLat(), locations[j].getLong())
                route = [start, end]
                routes.append(route)

    # SET UP MAP
    m = folium.Map(location=[(locations[0].getLat()+locations[-1].getLat())/2,
(locations[0].getLong()+locations[-1].getLong())/2], zoom_start=13)

    # VISUALIZE LOCATIONS
    for loc in locations:
        point_color = 'blue'
        if (loc == source_loc):
            point_color = 'red'
        if (loc == destination_loc):
            point_color = 'green'
        folium.Marker(location=[loc.getLat(), loc.getLong()], tooltip=loc.getName(),
icon=folium.Icon(color=point_color)).add_to(m)

    # VISUALIZE ROUTES
    for route in routes:
        route_color = 'green'
        if route in shortest_routes:
            route_color = 'blue'
```

```
        folium.PolyLine(locations=route, color=route_color).add_to(m)

    return m._repr_html_()

if __name__ == '__main__':
    try:
        result = main_program()
        app.run(debug=False)
    except:
        print("""Since the txt file didn't suit the format we use, you can use
another
            txt file by running the program again :)""")
```

parse.py

```
from datatype.Location import Location

def parse(filename):
    # ASSUMPTION : filename is defined and exist in tests directory
    # FILENAME FORMAT : name.txt
    filepath = 'tests/' + filename

    # CREATE LIST OF LOCATION
    locations = []

    # CREATE ADJACENCY MATRIX
    adjacency = []

    try:
        # OPEN FILE
        with open(filepath, 'r') as f:
            lines = f.readlines()

            # FIRST LINE : NUMBER OF VERTICES
            number_of_vertices = int(lines[0])

            # VERTICES LOCATION
            for line in range(1, 2 * number_of_vertices + 1, 2):
                # EXTRACT DATA
                loc_name = lines[line].strip()
                loc_latitude, loc_longtitude = map(float,
lines[line+1].strip().split(", "))

                # CREATE LOCATION
                location = Location(loc_name, loc_latitude, loc_longtitude)
```

```
                # APPEND TO LOCATION LIST
                locations.append(location)

            # ADJACENCY MATRIX
            for line in range(2 * number_of_vertices + 1, len(lines)):
                row = list(map(float, lines[line].split()))
                adjacency.append(row)

        # RETURN LOCATIONS AND ADJACENCY MATRIX
        return locations, adjacency
    except Exception as e:
        print(e)
```

haversine.py

```
from math import radians, sin, cos, sqrt, atan2
from datatype.Location import Location

def haversine(location1 : Location, location2 : Location):
    # GET LATITUDE AND LONGTITUDE
    lat1 = location1.getLat()
    long1 = location1.getLong()
    lat2 = location2.getLat()
    long2 = location2.getLong()

    # EARTH RADIUS (km)
    R = 6371

    # CONVERT LATITUDE AND LONGTITUDE TO RADIANS
    lat1, long1, lat2, long2 = map(radians, [lat1, long1, lat2, long2])

    # FIND DELTA OF LATITUDE AND LONGTITUDE
    dlat = lat2 - lat1
    dlon = long2 - long1

    # HAVERSINE FORMULA
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c

    return distance
```

ucs.py

```
from datatype.Graph import Graph
```

```
from utils.haversine import haversine
import heapq

def ucs(graph: Graph, src: str, dest: str):

    # Find all location neighbors
    graph.setNeighbour()

    # Get source and destination location
    srcLoc = graph.findLocation(src)
    destLoc = graph.findLocation(dest)

    # Initialize priority queue
    priorityqueue = []

    # Push source destination to priority queue
    heapq.heappush(priorityqueue, (0, srcLoc, srcLoc.name))

    # Variable for looping condition
    check = True

    # Looping until path from src to destination is found or until priority queue is
empty
    while check:
        # Dequeue priority queue tuple to check
        checkLocation = heapq.heappop(priorityqueue)

        # Return checkLocation if Location == destination
        if (checkLocation[1] == destLoc):
            return checkLocation

        # Adding all neighbor for current location to priority queue
        for neighborsName in checkLocation[1].neighbour:
            # Get neighbor location
            neighbors = graph.findLocation(neighborsName)

            # Get location index
            locationIndex = graph.findLocationIndex(checkLocation[1])

            # Get neighbor index
            neighborIndex = graph.findLocationIndex(neighbors)

            # Initialize new cost or weight for the neighbor location
            newWeight = checkLocation[0] +
graph.adjacencyMatrix[locationIndex][neighborIndex]
```

```
            # Initialize new path for the neighbor location
            newPath = checkLocation[2] + " - " + neighborsName

            # Remove current location from the neighbour's neighbor so there is no
loop between location
            if (checkLocation[1].name in neighbors.neighbour):
                neighbors.neighbour.remove(checkLocation[1].name)

            # Adding neighbor to prioqueue
            heapq.heappush(priorityqueue, (newWeight, neighbors, newPath))

        # Looping condition
        check = (len(priorityqueue) != 0 or checkLocation[1] != destLoc)
```

astar.py

```
from datatype.Graph import Graph
from datatype.Node import Node
from utils.haversine import haversine
import heapq

def astar(graph: Graph, src: str, dest: str):
    '''
    Function to find shortest path using A* algorithm
    graph : Graph of the map
    src: Source location name
    dest: Destination location name
    '''

    # FIND ALL LOCATIONS NEIGHBOUR
    graph.setNeighbour()

    # GET SOURCE AND DESTINATION LOCATION
    locSrc = graph.findLocation(src)
    locDest = graph.findLocation(dest)

    # INITIALIZE PRIORITY QUEUE
    prioqueue = []

    # PUSH THE SOURCE LOCATION TO PRIORITY QUEUE
    heapq.heappush(prioqueue, Node(locSrc, 0, haversine(locSrc, locDest),
locSrc.name))

    # SET BOOLEAN FOR CHECKING SO THAT IT WILL NOT CHECK THE SAME LOCATION TWICE
```

```
    checked = [0 for _ in range(len(graph.locations))]

    # SET BOOLEAN FOR CHECK CONDITION
    check = True

    while check:
        # DEQUEUE NODE TO CHECK
        checkNode = heapq.heappop(prioqueue)

        # CHECK IF CURRENT NODE IS DESTINATION
        if checkNode.loc == locDest:
            return checkNode

        # FIND CURRENT NODE INDEX
        indexCheckNode = graph.findLocationIndex(checkNode.loc)

        # SET CHECKED TO TRUE
        checked[indexCheckNode] = True

        # ITERATE ALL NEIGHBOUR IN CURRENT NODE
        for neighbour in checkNode.loc.neighbour:
            # GET LOCATION OF CURRENT NEIGHBOUR
            locToAdd = graph.findLocation(neighbour)

            # FIND THE INDEX LOCATION FOR CURRENT NEIGHBOUR
            indexToAdd = graph.findLocationIndex(locToAdd)

            # IF NEIGHBOUR HAS NEVER BEEN CHECKED
            if (not checked[indexToAdd]):
                # CREATE NODE TO BE ADDED TO PRIOQUEUE
                nodeToAdd = Node(locToAdd, haversine(checkNode.loc, locToAdd) +
checkNode.g, haversine(locToAdd, locDest), checkNode.route + " " + locToAdd.name)

                # ENQUEUE TO PRIOQUEUE
                heapq.heappush(prioqueue, nodeToAdd)

        # CHECK ENDING CONDITION
        check = (len(prioqueue) != 0 or checkNode.loc != locDest)
```

Graph.py

```
from datatype.Location import *
from utils.parse import *
from utils.haversine import *

class Graph:
```

```
    '''
    Class Graph for mapping the map
    '''

    # CONSTRUCTOR
    def __init__(self, filename):
        '''
        locations: list of Location in the map
        adjacencyMatrix: adjacency matrix of locations in map
        '''
        self.locations, self.adjacencyMatrix = parse(filename)

    # FIND LOCATION BY NAME
    def findLocation(self, name):
        for location in self.locations:
            if (location.getName() == name):
                return location

    # FIND LOCATION BY INDEX
    def findByIndex(self, index):
        return self.locations[index]

    # FIND INDEX LOCATION
    def findLocationIndex(self, location):
        for index in range(len(self.locations)):
            if (self.locations[index] == location):
                return index

    # SET NEIGHBOUR OF ALL LOCATIONS
    def setNeighbour(self):
        for i in range(len(self.locations)):
            for j in range(len(self.locations)):
                if (self.adjacencyMatrix[i][j] == 1):
                    self.locations[i].addNeighbour(self.locations[j].name)

    # Print names of all locations
    def printNames(self):
        for i in range(len(self.locations)):
            print(f'''[{i+1}] {self.locations[i].name}''')
```

Location.py

```
class Location:
    '''
    Class Location for each location in map
    '''
```

```
    # CONSTRUCTOR
    def __init__(self, name, latitude, longitude):
        '''
        name: name of the location
        latitude: latitude of the location
        longtitude: longtitude of the location
        neighbour: list of string, neighbour of the location
        '''
        self.name = name
        self.latitude = latitude
        self.longitude = longitude
        self.neighbour = []

    # STRING REPRESENTATION OF Location
    def __str__(self):
        return f"{self.name}, {self.latitude}, {self.longitude}"

    # GETTER
    def getName(self):
        return self.name

    def getLat(self):
        return self.latitude

    def getLong(self):
        return self.longitude

    # ADD NEIGHBOUR
    def addNeighbour(self, neighbour):
        self.neighbour.append(neighbour)
```

Node.py

```
class Node:
    '''
    Class Node to be used while finding shortest_path
    '''

    # CONSTRUCTOR
    def __init__(self, loc, g ,h, route):
        '''
        loc: Location of node
        g: distance from source to current node
        h: straight-line from current node to destination
        f: g + h
```

```
        route: route used from source to current node
        '''
        self.loc = loc
        self.g = g
        self.h = h
        self.f = g + h
        self.route = route

    # OVERRIDE LESS THAN OPERATOR
    def __lt__(self, other):
        return self.f < other.f

    # STRING REPRESENTATION OF Node
    def __str__(self):
        return f"{self.loc}, {self.g}, {self.h}, {self.f}, {self.route}"
```

BAB III

PENGUJIAN PROGRAM

3.1. Format Testcase

```
10
wtc_batanghari
-1.5886600997635014, 103.6158706067308
lippo_plaza
-1.596587479436691, 103.62609115518774
vihara_sakyakirti
-1.5973920275016154, 103.621233541694
sma3_jambi
-1.616102582895328, 103.62033225518779
bw_luxury
-1.618705858629114, 103.62972930916288
swissbel_hotel
-1.604248927128273, 103.6009734374142
jamtos
-1.6213659589370115, 103.58708197052935
smp_xaverius1
-1.625771986621938, 103.63119569130376
miepangsit_acai
-1.6085749397173346, 103.614100544824
nasi_amak
-1.6270631086160343, 103.62758482635236
0 1.4379110767491337 1.1393297510522387 0 0 2.3971855546524754 0 0 0 0
1.4379110767491337 0 0.5472934699129725 0 2.4924745102255432 0 0 0 0 0
1.1393297510522387 0.5472934699129725 0 0 0 0 0 1.4747371723989606 0
0 0 0 0 1.0838586436638076 2.5233583050115707 0 0 1.086465373381843 1.4612333050706974
0 2.4924745102255432 0 1.0838586436638076 0 0 0 0.8024447045933654 0 0.9593663794847801
2.3971855546524754 0 0 2.5233583050115707 0 0 2.450865249969488 0 1.5363417591860993 0
0 0 0 0 0 2.450865249969488 0 0 0 4.546255808378914
0 0 0 0 0 0.8024447045933654 0 0 0 0 0.4262529329617658
0 0 1.4747371723989606 1.086465373381843 0 1.5363417591860993 0 0 0 0
0 0 0 1.4612333050706974 0.9593663794847801 0 4.546255808378914 0.4262529329617658 0 0
```

Gambar 3.1.1 Contoh test case

Berikut penjelasan tentang format test case diatas :

- Pada line 1 akan diisi dengan jumlah simpul / lokasi pada graf / peta
- Dimulai dari line 2-21 diisi dengan nama_lokasi, dan latitude dan longitude lokasi tersebut dengan format berikut :
  nama_lokasi
  latitude_lokasi, longitude_lokasi
- Dimulai dari line 22-31 merupakan *adjacency matrix* berbobot
- Sebagai informasi tambahan pada test case kami diasumsikan akan selalu ada jalan menuju lokasi / simpul mana pun

3.2. Input Graf dan Peta

Tabel 3.2.1 Input graf dan peta

alunalunbandung.txt

```
10
AlunAlunBandung
-6.921825592497l9, 107.60695420607672
MuseumAsiaAfrika
-6.921189604928431, 107.60952504581432
PLNBandung
-6.920880652884007, 107.60837577208487
Cikapundung
-6.919081870809035, 107.60892135022254
KantorPos
-6.920667768326371, 107.60619855988753
MasjidRayaBandung
-6.921681840605713, 107.60607795650104
PendopoBandung
-6.9233941110536215, 107.60698555328437
BankPanin
-6.919826236345531, 107.60679326883204
GrandYogyaKepatihan
-6.923618925963251, 107.60572256674298
Warung_SPG
-6.925124354291612, 107.60728887688899
0 0 0.1888486075283205 0 0 0.09803625668328939 0.17444562722151197 0 0 0
0 0 0.13143138130040882 0.24365903668129396 0 0 0 0 0 0.502350698467662
0.1888486075283205 0.13143138130040882 0 0.20888527499335954
0.24149394971297594 0 0 0.2103834406220895 0 0
0 0.24365903668129396 0.20888527499335954 0 0 0 0 0.24906374445709775 0 0
0 0 0.24149394971297594 0 0 0.11354284610660265 0 0.11430494228786688 0 0
0.09803625668328939 0 0 0 0.11354284610660265 0 0 0 0.21893732517845632 0
0.17444562722151197 0 0 0 0 0 0 0 0.14163711133211235 0.1952859507160381
0 0 0.2103834406220895 0.24906374445709775 0.11430494228786688 0 0 0 0 0
0 0 0 0 0 0.21893732517845632 0.14163711133211235 0 0 0
0 0.502350698467662 0 0 0 0 0.1952859507160381 0 0 0
```

buahbatu.txt

```
12
margacinta_park
-6.954752360856165, 107.64794103590319
perumahan_kiara
-6.948848323689184, 107.64551337730865
kembang_residence
-6.946286072674258, 107.65224652138247
metroindah_mall
-6.941834840986075, 107.65875736831948
bolu_banjur
-6.94820989706843, 107.65844069315717
RSIA_Harapan
-6.955729082930565, 107.66197479086469
sman21_bandung
-6.956722411247707, 107.67131038284772
buana_ciwastra
-6.9584536020525025, 107.66833060741143
smp_pgri7
-6.959333761332411, 107.65193948684113
komplek_megabrata
```

```
-6.957746507380969, 107.65808931184885
smkn10_bandung
-6.963990264675813, 107.6579648721037
matahari_margawangi
-6.963282400536106, 107.65417165595888
0 0.7090788529200337 1.0545581875000545 0 0 0 0 0 0.6740126237354365
1.168561869286041 0 0
0.7090788529200337 0 0.7959338302822457 0 0 0 0 0 0 0 0 0
1.0545581875000545 0.7959338302822457 0 0.8726142991772274 0.716388236372206 0
0 0 0 0 0 0
0 0 0.8726142991772274 0 0.7097351619428125 0 2.1587605860505583 0 0 0 0 0
0 0 0.716388236372206 0.7097351619428125 0 0.9226165850333083 0 0 0
1.0611317094412707 0 0
0 0 0 0 0.9226165850333083 0 0 0.7641505345265002 0 0.4839914991397054 0 0
0 0 0 2.1587605860505583 0 1.0363320963357967 0 0.38108880497169695 0 0 0 0
0 0 0 0 0 0.7641505345265002 0.38108880497169695 0 0 1.1331241259427407
1.2992441159619101 0
0.6740126237354365 0 0 0 0 0 0 0 0 0.7013624847281358 0.842846844651523
0.5034704461554588
1.168561869286041 0 0 0 1.0611317094412707 0.4839914991397054 0
1.1331241259427407 0.7013624847281358 0 0.6944099841089553 0.7522625777373955
0 0 0 0 0 0 0 1.2992441159619101 0 0.6944099841089553 0 0.4260095776984032
0 0 0 0 0 0 0 0 0 0.5034704461554588 0.7522625777373955 0.4260095776984032 0
```

itb.txt

```
14
ParkirSipil
-6.893080, 107.608640
Kubus
-6.893225341866101, 107.6104810985063
CIBE
-6.891197, 107.608606
LabtekV
-6.890610, 107.610071
Oktagon
-6.889082, 107.610163
Plazwid
-6.889846277776866, 107.61034392232239
AulaTimur
-6.892443757589303, 107.61103842494695
GedungArsi
-6.891824973040149, 107.61187877908219
GKUTimur
-6.890369, 107.611841
GKUBarat
-6.890237, 107.608724
CADL
-6.888400976370087, 107.6094940681372
PerpustakaanPusat
-6.888547699076628, 107.61075509832881
CRCS
-6.887907780586676, 107.61160336176233
MasjidSalman
-6.8936779899060445, 107.61140864115396
0 0.2038825703609363 0.20941368472222527 0 0 0 0.27405423146252283 0 0 0 0 0 0
0
0.2038825703609363 0 0 0.2943153939315096 0 0 0.10648114505735144 0 0 0 0 0 0
0.11409444539726293
0.20941368472222527 0 0 0.17439884967960645 0 0 0 0 0 0.10753898003843236 0 0
0 0
```

```
0 0.2943153939315096 0.17439884967960645 0 0 0.09010811780854355 0 0
0.19722257216402184 0.15437377035303887 0 0 0 0
0 0 0 0 0 0.08729916540891876 0 0 0 0 0.10577120169522561 0.0883292590998183 0
0
0 0 0 0.09010811780854355 0.08729916540891876 0 0 0 0.17518847352671527
0.18402834054431735 0 0 0 0
0.27405423146252283 0.10648114505735144 0 0 0 0 0 0.1154993692007701 0 0 0 0 0
0
0 0 0 0 0 0 0.1154993692007701 0 0.1619505228981837 0 0 0 0
0.21248177267571292
0 0 0 0.19722257216402184 0 0.17518847352671527 0 0.1619505228981837 0 0 0 0
0.2749295501812586 0
0 0 0.10753898003843236 0.15437377035303887 0 0.18402834054431735 0 0 0 0
0.2211481002824896 0 0 0
0 0 0 0 0.10577120169522561 0 0 0 0 0.2211481002824896 0 0.14016074608777712 0
0
0 0 0 0 0.0883292590998183 0 0 0 0 0 0.14016074608777712 0 0.11760915480411586
0
0 0 0 0 0 0 0 0 0.2749295501812586 0 0 0.11760915480411586 0 0
0 0.11409444539726293 0 0 0 0 0 0.21248177267571292 0 0 0 0 0 0
```

jambi.txt

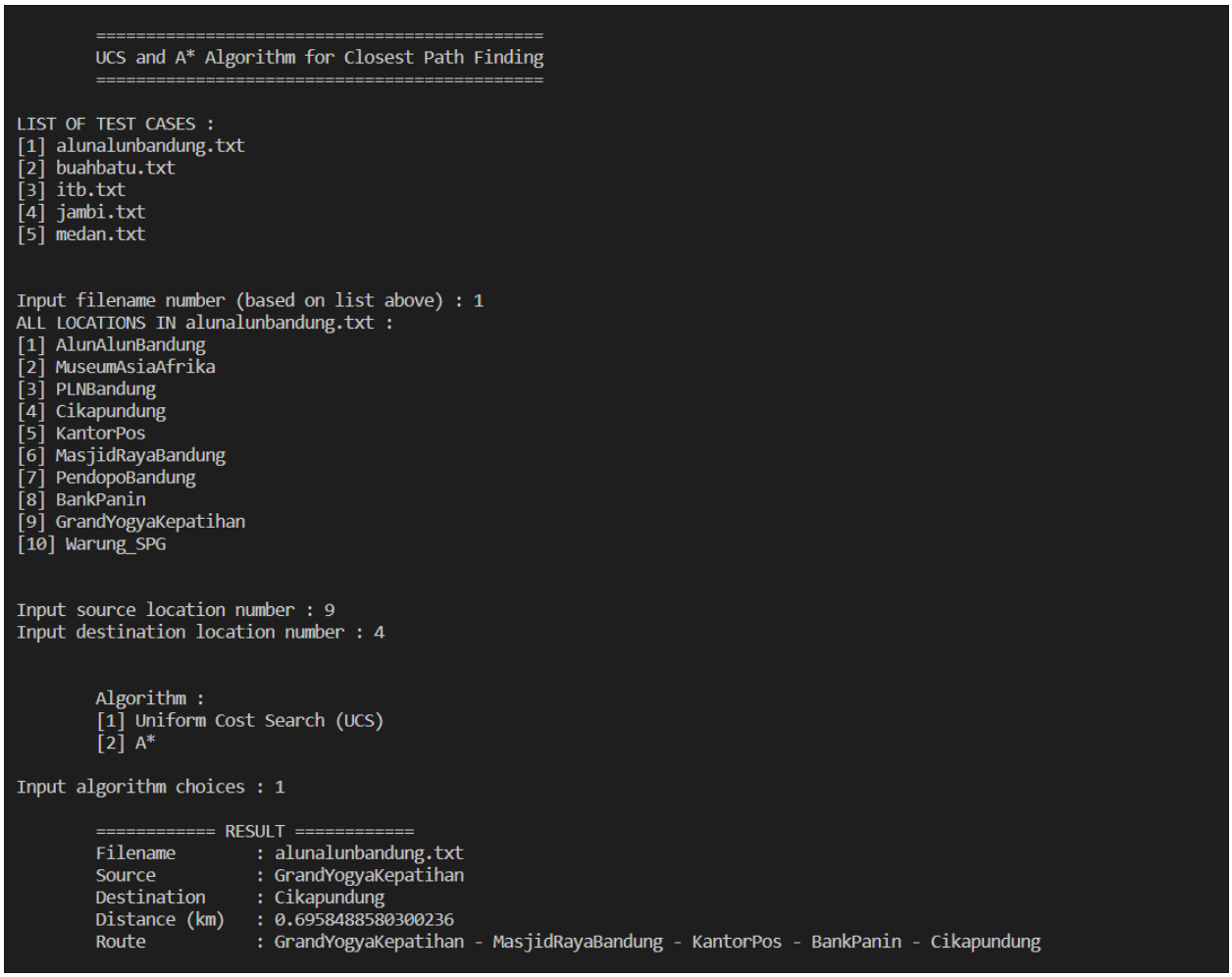
```
10
wtc_batanghari
-1.5886600997635014, 103.6158706067308
lippo_plaza
-1.596587479436691, 103.62609115518774
vihara_sakyakirti
-1.5973920275016154, 103.621233541694
sma3_jambi
-1.616102582895328, 103.62033225518779
bw_luxury
-1.618705858629114, 103.62972930916288
swissbel_hotel
-1.604248927128273, 103.6009734374142
jamtos
-1.6213659589370115, 103.58708197052935
smp_xaverius1
-1.625771986621938, 103.63119569130376
miepangsit_acai
-1.6085749397173346, 103.614100544824
nasi_amak
-1.6270631086160343, 103.62758482635236
0 1.4379110767491337 1.1393297510522387 0 0 2.3971855546524754 0 0 0 0
1.4379110767491337 0 0.5472934699129725 0 2.4924745102255432 0 0 0 0 0
1.1393297510522387 0.5472934699129725 0 0 0 0 0 0 1.4747371723989606 0
0 0 0 0 1.0838586436638076 2.5233583050115707 0 0 1.086465373381843
1.4612333050706974
0 2.4924745102255432 0 1.0838586436638076 0 0 0 0.8024447045933654 0
0.9593663794847801
2.3971855546524754 0 0 2.5233583050115707 0 0 2.450865249969488 0
1.5363417591860993 0
0 0 0 0 0 2.450865249969488 0 0 0 4.546255808378914
0 0 0 0 0.8024447045933654 0 0 0 0 0.4262529329617658
0 0 1.4747371723989606 1.086465373381843 0 1.5363417591860993 0 0 0 0
0 0 0 1.4612333050706974 0.9593663794847801 0 4.546255808378914
0.4262529329617658 0 0
```

medan.txt

```
10
delipark_mall
3.594471749913244, 98.67445484758547
bolu_meranti
3.594304535666858, 98.66641274479056
centre_point_mall
3.5915622179030184, 98.6809555468108
carrefour
3.5917294325031843, 98.66363151625788
ucok_durian
3.58269979285585, 98.65699678182092
pondok_gurih
3.580117832553302, 98.68344487446511
tabona
3.5817554955185105, 98.68150301393518
methodist_3
3.5991756765391703, 98.68780912399936
merdeka_walk
3.59041791728566, 98.67856897247421
sun_plaza
3.5825253838719107, 98.6715003457089
0 0.8926756052881467 0.790645964930403 1.239235966217944 0 0 0
1.5715967886631892 0.6415969297247587 0
0.8926756052881467 0 0 0.42101649172515093 0 0 0 0 0 0
0.790645964930403 0 0 0 0 0 0 1.1380601432128925 0.2938328154256705 0
1.239235966217944 0.42101649172515093 0 0 1.2450940769196068 0 0 0 0 0
0 0 0 1.2450940769196068 0 0 0 0 0 0
0 0 0 0 0 0 0.2821384111708344 0 0 0
0 0 0 0 0 0.2821384111708344 0 0 1.016764797743311 0
1.5715967886631892 0 1.1380601432128925 0 0 0 0 0 0 0
0.6415969297247587 0 0.2938328154256705 0 0 0 1.016764797743311 0 0
1.177102410792859
0 0 0 0 0 0 0 0 1.177102410792859 0
```

3.3. *Screenshot* Program Utama pada CLI

Tampilan utama program ditunjukkan oleh gambar dibawah ini :



Gambar 3.3.1 Tampilan Utama Program

3.4. *Screenshot* Hasil Program Menggunakan Algoritma UCS

Tabel 3.4.1 Test case dan hasil pada CLI menggunakan algoritma UCS

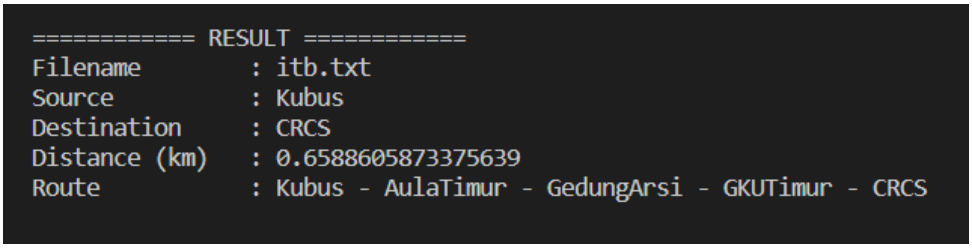
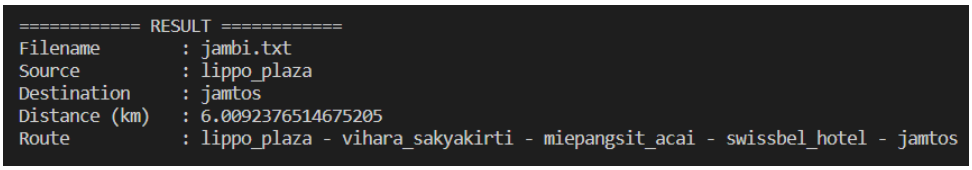
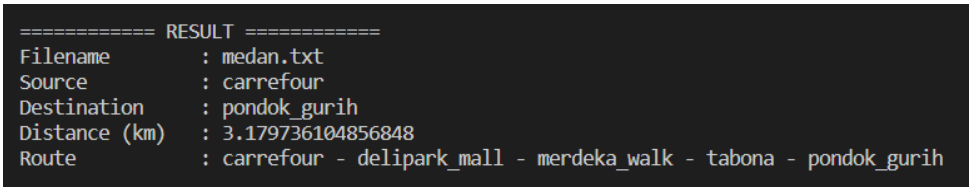
Test Case	Hasil
alunalunbandung.txt Start : GrandYogyaKepatihan End : Cikapundung	========== RESULT ========== Filename : alunalunbandung.txt Source : GrandYogyaKepatihan Destination : Cikapundung Distance (km) : 0.6958488580300236 Route : GrandYogyaKepatihan - MasjidRayaBandung - KantorPos - BankPanin - Cikapundung

buahbatu.txt Start : metroindah_mall End : smp_pgri7	``` ============ RESULT ============ Filename : buahbatu.txt Source : metroindah_mall Destination : smp_pgri7 Distance (km) : 2.472229356112219 Route : metroindah_mall - bolu_banjur - komplek_megabrata - smp_pgri7 ```
itb.txt Start : Kubus End : CRCS	``` ============ RESULT ============ Filename : itb.txt Source : Kubus Destination : CRCS Distance (km) : 0.6588605873375639 Route : Kubus - AulaTimur - GedungArsi - GKUTimur - CRCS ```
jambi.txt Start : lippo_plaza End : jamtos	``` ============ RESULT ============ Filename : jambi.txt Source : lippo_plaza Destination : jamtos Distance (km) : 6.0092376514675205 Route : lippo_plaza - vihara_sakyakirti - miepangsit_acai - swissbel_hotel - jamtos ```
medan.txt Start : carrefour End : pondok_gurih	``` ============ RESULT ============ Filename : medan.txt Source : carrefour Destination : pondok_gurih Distance (km) : 3.179736104856848 Route : carrefour - delipark_mall - merdeka_walk - tabona - pondok_gurih ```

3.5. *Screenshot* Hasil Program Menggunakan Algoritma A*

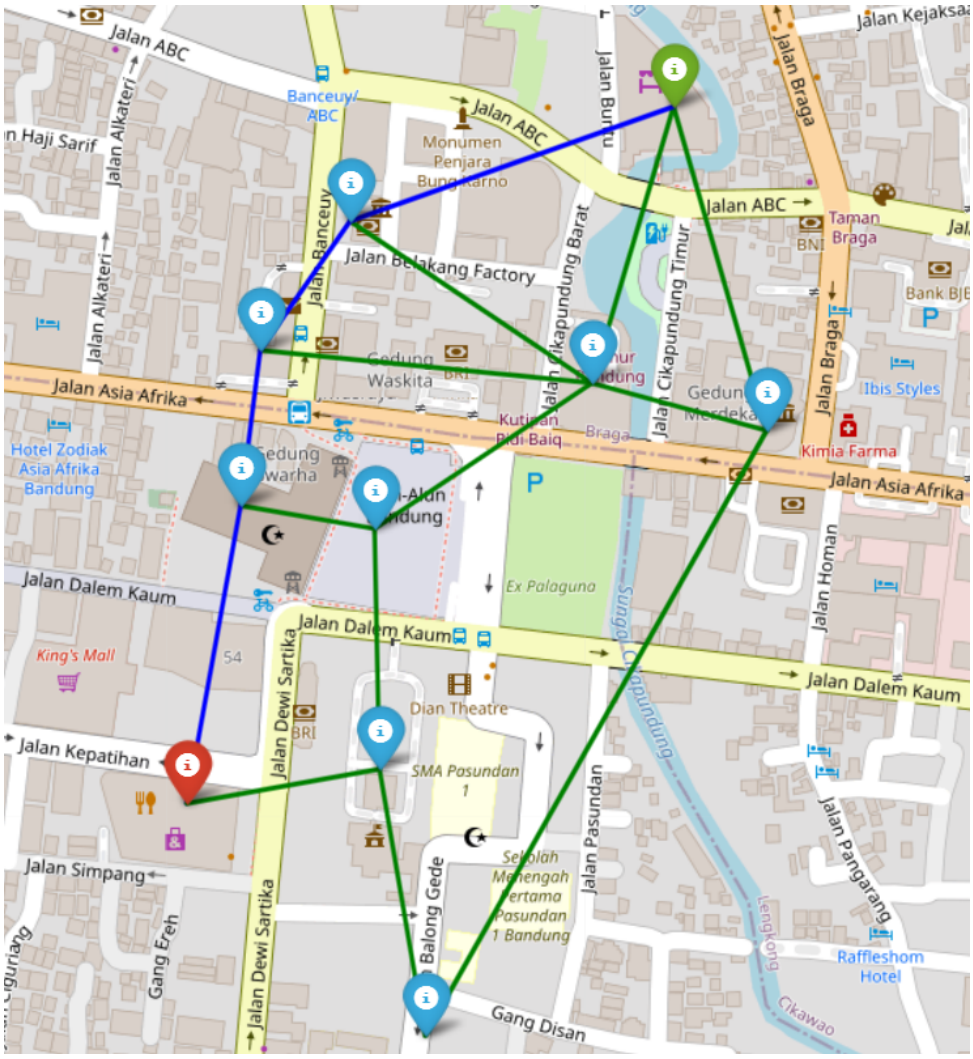
Tabel 3.5.1 Test case dan hasil pada CLI menggunakan algoritma A*

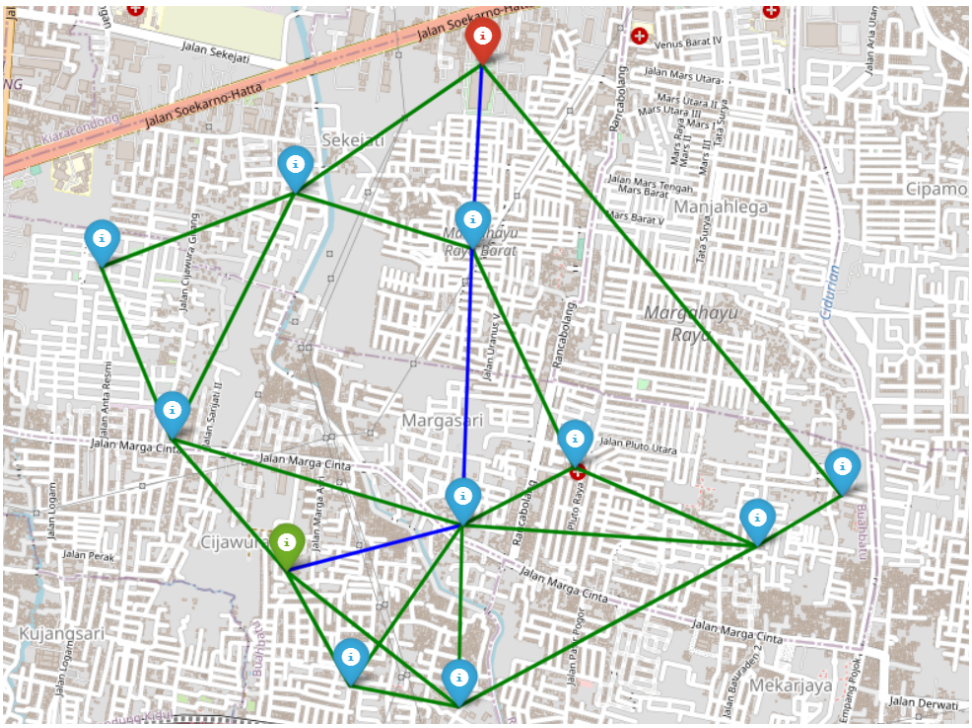
Test Case	Hasil
alunalunbandung.txt Start : GrandYogyaKepatihan End : Cikapundung	``` ============ RESULT ============ Filename : alunalunbandung.txt Source : GrandYogyaKepatihan Destination : Cikapundung Distance (km) : 0.6958488580300236 Route : GrandYogyaKepatihan - MasjidRayaBandung - KantorPos - BankPanin - Cikapundung ```
buahbatu.txt Start : metroindah_mall End : smp_pgri7	``` ============ RESULT ============ Filename : buahbatu.txt Source : metroindah_mall Destination : smp_pgri7 Distance (km) : 2.472229356112219 Route : metroindah_mall - bolu_banjur - komplek_megabrata - smp_pgri7 ```

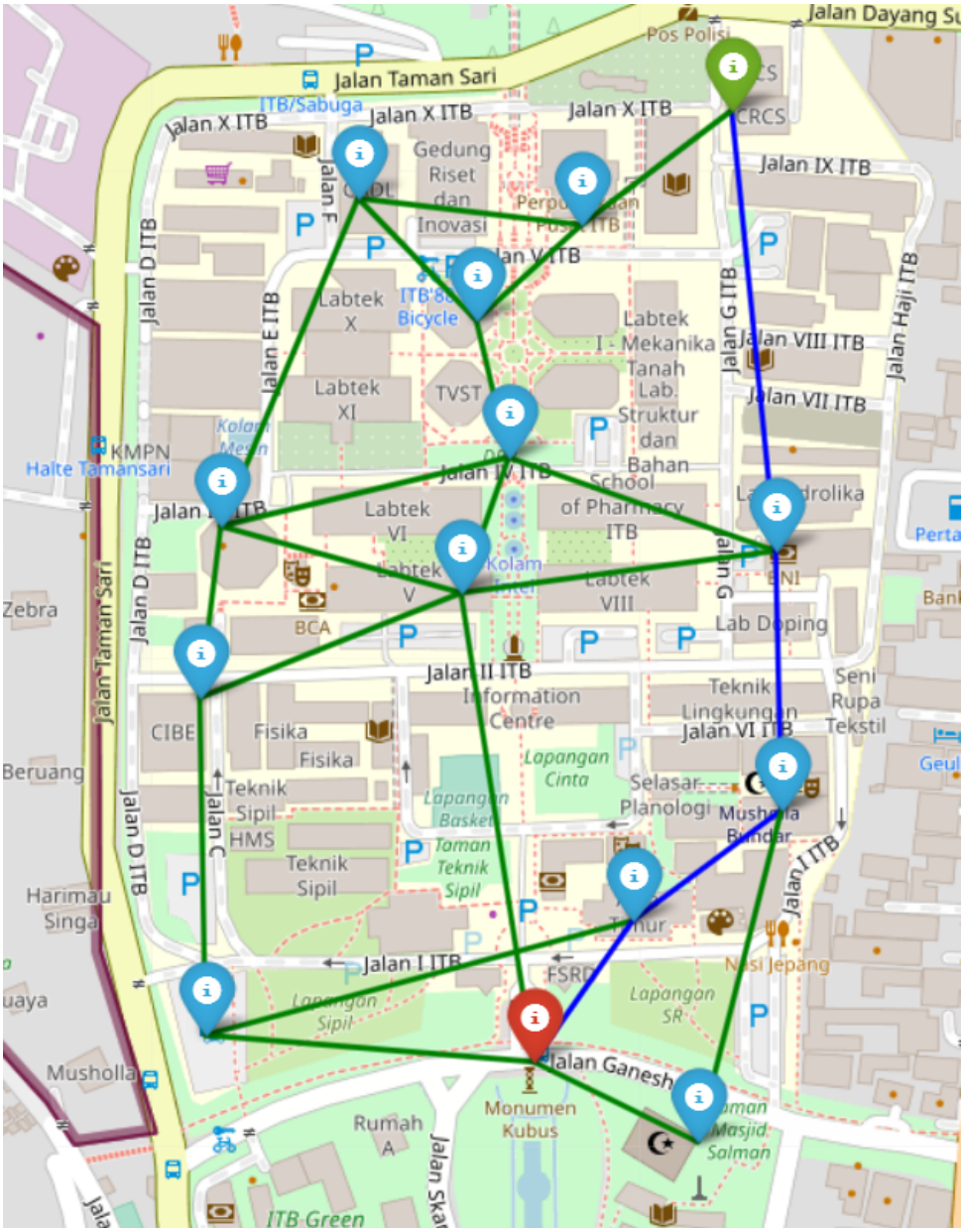
itb.txt Start : Kubus End : CRCS	
jambi.txt Start : lippo_plaza End : jamtos	
medan.txt Start : carrefour End : pondok_gurih	

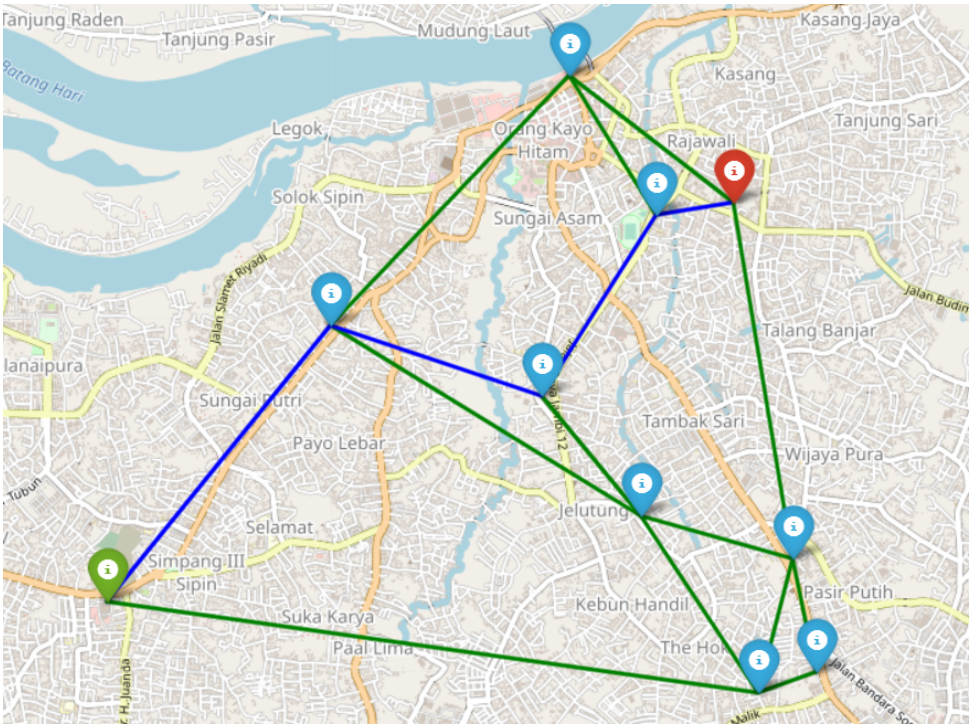
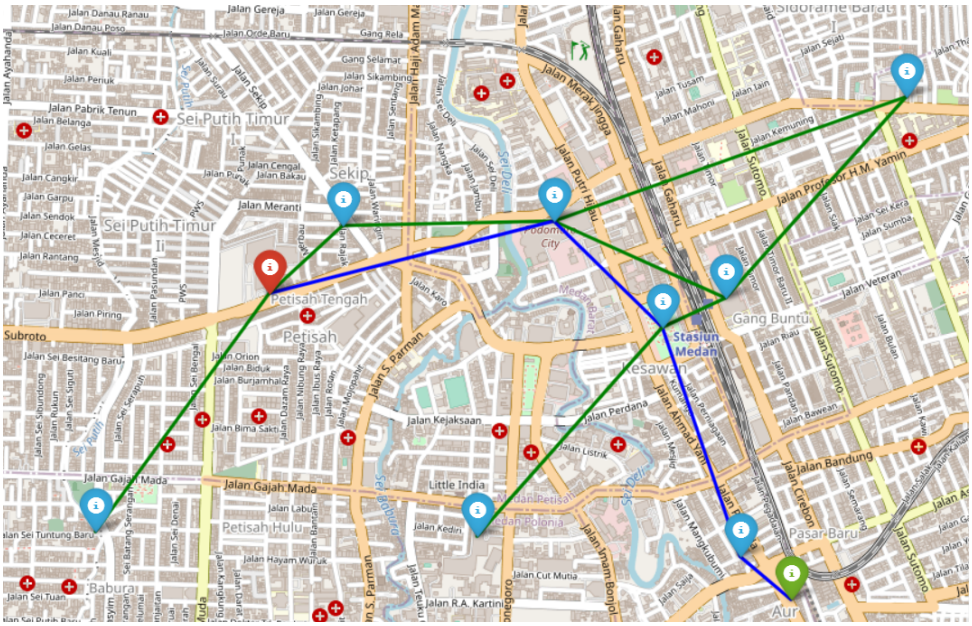
3.6. *Screenshot* Hasil Visualisasi pada Peta

Tabel 3.6.1 Test case dan hasil visualisasi pada peta

Test Case	Visualisasi pada peta
alunalunbandung.txt Start : GrandYogyaKepatihan End : Cikapundung	

buahbatu.txt Start : metroindah_mall End : smp_pgri7	

itb.txt Start : Kubus End : CRCS	

jambi.txt Start : lippo_plaza End : jamtos	
medan.txt Start : carrefour End : pondok_gurih	

BAB IV

KESIMPULAN

Dengan algoritma *Uniform Cost Search* (UCS) dan algoritma *A* Search* untuk lintasan terpendek antara 1 titik pada *google maps* dengan 1 titik lainnya dapat ditemukan. Hal ini dikarenakan algoritma *Uniform Cost Search* dan *A* Search* sudah memberikan solusi lintasan terpendek yang optimal. Dapat dilihat berdasarkan percobaan yang telah dilakukan diatas, algoritma *Uniform Cost Search* dan *A* Search* menghasilkan rute yang sama dan solusi yang sudah optimal.

LAMPIRAN

- **Github Repository**

  https://github.com/MarcelRyan/Tucil3_13521046_13521127.git

- ***Checklist* program**

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 (tidak diimplementasi) dan menampilkan peta serta lintasan terpendek pada peta (diimplementasi)	✓	✓