Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II Tahun 2022/2023

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun oleh:

Azmi Hasna Zahrani	13521006
Kenny Benaya Nathan	13521023

K3

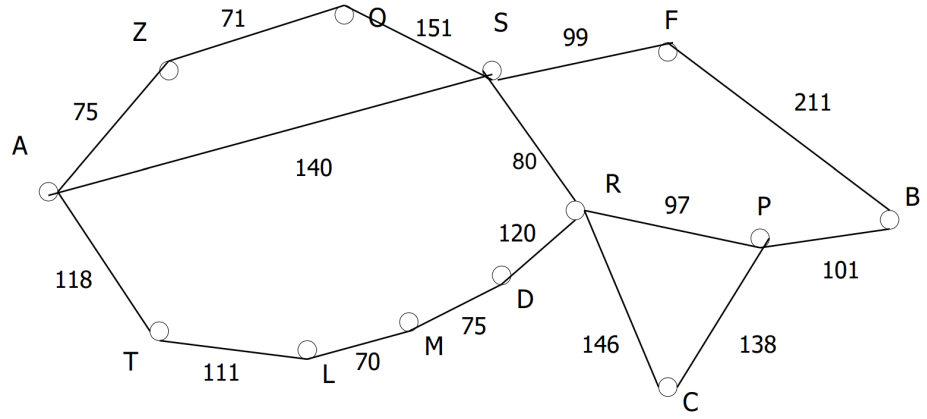
TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

1.1 Algoritma UCS

Uniform-Cost Search (UCS) merupakan salah satu algoritma pencarian yang menggunakan total cost terendah untuk mencari *path* dari *start node* hingga *goal node.* Algoritma ini termasuk algoritma pencarian blind search karena bekerja secara brute force. UCS tidak memasukkan semua simpul ke dalam priority queue, melainkan hanya menyisipkan node sumber lalu memasukkan satu persatu bila diperlukan. Pada setiap iterasi, apabila node telah ada dalam priority queue maka akan dilakukan kunci penurunan. Sebaliknya, apabila node belum ada di priority queue, maka node akan dimasukkan ke dalam antrian. Contoh penerapan algoritma UCS adalah sebagai berikut.



Gambar 1.1.1 Weighted Graph
Sumber: Bahan Kuliah IF2211 Strategi Algoritma: Penentuan Rute Bagian 1

Simpul-E	Simpul Hidup
A	Z_{A-75}, T_{A-118}, S_{A-140}
Z_{A-75}	T_{A-118}, S_{A-140}, O_{AZ-146}
T_{A-118}	S_{A-140}, O_{AZ-146}, L_{AT-229}
A_{A-140}	O_{AZ-146}, R_{AS-220}, L_{AT-229}, F_{AS-239}, O_{AS-291}
O_{AZ-146}	R_{AS-220}, L_{AT-229}, F_{AS-239}, O_{AS-291}

R_{AS-220}	L_{AT-229}, F_{AS-239}, O_{AS-291}, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$
L_{AT-229}	F_{AS-239}, O_{AS-291}, $M_{ASL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$
F_{AS-239}	O_{AS-291}, $M_{ASL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$, $B_{ASF-450}$
O_{AS-291}	$M_{ASL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$, $B_{ASF-450}$
$M_{ASL-299}$	$P_{ASR-317}$, $D_{ASR-340}$, $D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASF-450}$
$P_{ASR-317}$	$D_{ASR-340}$, $D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$D_{ASR-340}$,	$D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$D_{ATLM-364}$	$C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$C_{ASR-366}$	$B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$B_{ASRP-418}$	Solusi Ketemu

Tabel 1.1.2 Tabel Penyelesaian UCS

1.2 Algoritma A*

Algoritma A* merupakan algoritma pencarian yang digunakan untuk menemukan jalur terpendek di antara *start point* dan *goal point.* Algoritma A* merupakan algoritma yang dapat menemukan hasil optimal karena mencari jalur yang lebih pendek terlebih dahulu. Pencarian ini menggunakan fungsi heuristik jarak ditambah biaya yang dinotasikan dalam fungsi sebagai berikut.

$$f(n) = g(n) + h(n)$$

dimana

f(n) = biaya estimasi terendah

g(n) = biaya dari node awal ke node n

h(n) = perkiraan biaya dari node n ke node akhir.

Algoritma A* mirip dengan algoritma Dijkstra, tetapi lebih modern karena mempertimbangkan biaya setiap edge dalam graph. Biasanya algoritma ini digunakan dalam peta dan game berbasis web untuk menemukan jalur terpendek dengan efisiensi setinggi mungkin. Selain itu, A* juga digunakan di banyak AI serta protokol routing jaringan.

1.3 Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Pencarian lintasan terpendek berdasarkan peta Google Map dapat dilakukan dengan mengimplementasikan algoritma UCS serta algoritma A*. Dari peta Google Map, terdapat ruas-ruas jalan yang dapat dibentuk graf, yaitu node (persimpangan atau ujung jalan) serta edge (jalan). Bobot graf merupakan jarak antara simpul dan jarak tersebut dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean.

Program yang dibuat oleh penulis akan meminta pilihan input yang digunakan, yaitu dapat berupa input file (.txt) atau menggunakan peta yang telah disediakan. Peta yang disediakan antara lain peta jalan sekitar kampus ITB Ganesha, peta jalan sekitar kampus ITB Jatinangor, peta jalan sekitar alun-alun Bandung, peta jalan sekitar Bandung Selatan, peta jalan di sekitar alun-alun Cilacap. Pengguna dapat melihat lokasi yang tersedia dengan menekan tombol “show”. Khusus untuk input berupa file (.txt), pengguna harus memasukkan file dengan menggunakan tombol “input file”. Kemudian, program akan meminta input *start node*, *goal node,* dan algoritma yang digunakan. Secara otomatis, graph akan ditampilkan pada antarmuka dan apabila input yang digunakan merupakan peta yang telah disediakan, maka hasil pencarian dapat dibuka dalam Google Map dengan menekan tombol “Open GMaps”.

BAB II
SOURCE CODE PROGRAM DALAM BAHASA PYTHON

2.1 algorithm.py

```
import heapq
from src.core import Area, Components, function

# UNIFORM COST SEARCH
# ======================================================================
def uniform_cost_search(graph: Components.Graph, start, goal):
    pqueue = [(0, (start, []))]      # queue of tuple (cost: int,
(nodename: string, path: list of node name))
    explored = set()                 # set of string (explored node)

    while pqueue:
        cost, (current, tempPath) = heapq.heappop(pqueue)
        # check if current node is goal
        if current == goal:
            return cost, (tempPath + [current])

        # check if current node is explored
        if current in explored:
            continue

        # add current node to explored set
        explored.add(current)

        # add adjacent node to queue
        currentNode = graph.getNode(current)
        for i in range(len(currentNode.getAdjacents())):
            # check if node is a neighbor
            if currentNode.getAdjacents()[i] == 0:
                continue

            new_cost = cost + currentNode.getAdjacents()[i]
            new_path = tempPath + [current]
            new_node = graph.getNodeByIdx(i).getName()

            heapq.heappush(pqueue, (new_cost, (new_node, new_path)))
```

```
    return 0, []


# A* SEARCH
================================================================================
=======
# Heuristic Function
# /*
#    h(n) = edge(s) covered to goal node
#    Covering all node using BFS
# */
def heuristic(graph: Components.Graph, goal: str):
    explored = set()                # set of string (explored node)
    value = 0                       # int (distance to goal (edge(s)
covered))
    queueNode = [(goal, value)]     # queue of string (node name)
    explored.add(goal)

    while queueNode:
        current, value = queueNode.pop(0)
        currentNode = graph.getNode(current)
        currentNode.setHeuristic(value)

        # add adjacent node to queue
        for i in range(len(currentNode.getAdjacents())):
            if currentNode.getAdjacents()[i] > 0 and
graph.getNodeByIdx(i).getName() not in explored:
                neighbor = graph.getNodeByIdx(i).getName()
                queueNode.append((neighbor, value + 1))
                explored.add(neighbor)

def heuristicMap(graph: Components.Graph, goal: str, place: Area.Area):
    idxGoal = graph.getIdxNode(goal)
    for i in range(len(graph.getMatrix())):
        if graph.getNameNode(i) == goal:
            graph.getNodeByIdx(i).setHeuristic(0)
            continue
        coorGoal = place.getListCoordinate()[idxGoal]
        coorI = place.getListCoordinate()[i]
```

```
        hn = function.haversineDistance(coorGoal[0], coorGoal[1],
coorI[0], coorI[1])
        graph.getNodeByIdx(i).setHeuristic(hn)

# Main Algorithm
def a_star(graph: Components.Graph, start: str, goal: str):
    # heuristic(graph, goal)
# set heuristic value for each node

    pqueue = [(0 + graph.getNode(start).getHeuristic(), (start, 0, []))]
# queue of tuple (f(n): int, (nodename: string, g(n): int, path: list of
node name))
    explored = set()
# set of string (explored node)

    while pqueue:
        fValue, (current, cost, tempPath) = heapq.heappop(pqueue)

        # check if current node is goal
        if current == goal:
            return cost, (tempPath + [current])

        # check if current node is explored
        if current in explored:
            continue

        # add current node to explored set
        explored.add(current)

        # add adjacent node to queue
        currentNode = graph.getNode(current)
        for i in range(len(currentNode.getAdjacents())):
            # check if node is a neighbor
            if currentNode.getAdjacents()[i] == 0:
                continue

            new_cost    = cost + currentNode.getAdjacents()[i]
            new_fValue  = new_cost + graph.getNodeByIdx(i).getHeuristic()
            new_path    = tempPath + [current]
            new_node    = graph.getNodeByIdx(i).getName()
```

```
            heapq.heappush(pqueue, (new_fValue, (new_node, new_cost,
new_path)))

    return 0, []
```

2.2 area.py

```
class Area:
    # ATRIBUTTES ----------------------------
    # center: tuple (x, y) center coordinate of the map
    # zoom: Int (zoom level)
    # listCoordinate: list of tuples [(x, y)] coordinate of nodes
    # listName: list of strings (name of nodes)
    # matrix: list of list of int (adjacency matrix)

    # CONSTRUCTOR ---------------------------
    def __init__(self, x, y, zoom, listCoordinate, listName, matrix):
        self.__center = (x, y)
        self.__zoom = zoom
        self.__listCoordinate = listCoordinate
        self.__listName = listName
        self.__matrix = matrix

    # GETTER ---------------------------------
    def getCenter(self):
        return self.__center

    def getZoom(self):
        return self.__zoom

    def getListCoordinate(self):
        return self.__listCoordinate

    def getListName(self):
        return self.__listName

    def getMatrix(self):
        return self.__matrix
```

2.3 Components.py

```
from src.core import function as func

class Node:
    # ATRIBUTTES ----------------------------
    # name: String
    # adjacents: List of Int (adjacency list)
    # heuristic: Int (heuristic value)

    # CONSTRUCTOR ----------------------------
    def __init__(self, name, adjacents):
        self.__name = name
        self.__adjacents = adjacents
        self.__heuristic = 0

    # GETTER ---------------------------------
    def getName(self):
        return self.__name

    def getAdjacents(self):
        return self.__adjacents

    def getHeuristic(self):
        return self.__heuristic

    def setHeuristic(self, heuristic):
        self.__heuristic = heuristic

    def getWeight(self, idx):
        return self.__adjacents[idx]

    # METHODS --------------------------------
    def __str__(self):
        adjacent_str = ''
        adjacent_node = ''
        for i in range(len(self.__adjacents)):
            adjacent_str += str(self.__adjacents[i]) + ' '
            if self.__adjacents[i] != 0:
                adjacent_node += str(i) + ' '
```

```
        return 'Name: ' + self.__name + '\nAdjacency List: ' +
adjacent_str + '\nAdjacent Node: ' + adjacent_node + '\nHeuristic: ' +
str(self.__heuristic) + '\n'


class Graph:
    # ATTRIBUTES ----------------------------
    # matrix: List of List of Int
    # nodes: List of Node

    # CONSTRUCTOR ---------------------------
    def __init__(self, matrix, nodes):
        self.__matrix = matrix
        self.__nodes = nodes

    # GETTER --------------------------------
    def getMatrix(self):
        return self.__matrix

    def getListNode(self):
        return self.__nodes

    def getNode(self, name):
        for i in range(len(self.__nodes)):
            if self.__nodes[i].getName() == name:
                return self.__nodes[i]
        return None

    def getNodeByIdx(self, idx):
        return self.__nodes[idx]

    def getIdxNode(self, node):
        for i in range(len(self.__nodes)):
            if self.__nodes[i] == node:
                return i
        return -1

    def getNeighbor(self, node):
        neighbor = []
```

```
        for i in range(len(self.__nodes)):
            if self.__matrix[self.getIdxNode(node)][i] != 0:
                neighbor.append(self.getNodeIdx(i))
        return neighbor

    def getNodeWeight(self, node1, node2):
        return
self.__matrix[self.getIdxNode(node1)][self.getIdxNode(node2)]

    def getNameNode(self, idx):
        return self.__nodes[idx].getName()

    # METHODS --------------------------------
    def convertCoordinatesToWAM(self, listCoordinate: list[list[int]]):
        for i in range(len(self.__matrix)):
            for j in range(len(self.__matrix[i])):
                if self.__matrix[i][j] == 1:
                    self.__matrix[i][j] =
func.haversineDistance(listCoordinate[i][0], listCoordinate[i][1],
listCoordinate[j][0], listCoordinate[j][1])
                    self.__matrix[j][i] = self.__matrix[i][j]
```

2.4 function.py

```
import numpy as np
from src.core import Components as comp


# Initiate list of node
# /* listNodeName: list of string (node name)
#    adjacentyMatrix: list of list of int (adjacency matrix)
#    return: list of Node
# */
def initiateListNode(listNodeName, adjacentyMatrix):
    listNode = []                                   # list of node
    for i in range(len(listNodeName)):              # iterate through list of
node name to iniate list of Node
        name = listNodeName[i]
        adjacentList = adjacentyMatrix[i]
        node = comp.Node(name, adjacentList)
        listNode.append(node)
```

```
    return listNode


# Count euclidean distance between two points
# euclidean distance is for 2D plane
def euclideanDistance(x1, y1, x2, y2):
    return np.sqrt(np.square(x1 - x2) + np.square(y1 - y2))


# Count haversine distance between two points
# /* haversine distance is for earth surface (sphere)
#    lat and lon are in degrees
#    return: approximate distance in kilometers
# */
def haversineDistance(lat1, lon1, lat2, lon2):
    R = 6372.8 # Earth radius in kilometers

    # convert lat and lon difference to radians
    dLat = np.radians(lat2 - lat1)
    dLon = np.radians(lon2 - lon1)

    # convert lat to radians
    lat1 = np.radians(lat1)
    lat2 = np.radians(lat2)

    # haversine formula
    a = np.sin(dLat/2)**2 + np.sin(dLon/2)**2 * np.cos(lat1) *
np.cos(lat2)
    c = 2 * np.arcsin(np.sqrt(a))

    # calculate final approximate distance in kilometers
    return R * c
```

2.5 function.py

```
import numpy as np
from src.core import Components as comp


# Initiate list of node
```

```
# /* listNodeName: list of string (node name)
#    adjacentyMatrix: list of list of int (adjacency matrix)
#    return: list of Node
# */
def initiateListNode(listNodeName, adjacentyMatrix):
    listNode = []                                   # list of node
    for i in range(len(listNodeName)):              # iterate through list of
node name to iniate list of Node
        name = listNodeName[i]
        adjacentList = adjacentyMatrix[i]
        node = comp.Node(name, adjacentList)
        listNode.append(node)

    return listNode


# Count euclidean distance between two points
# euclidean distance is for 2D plane
def euclideanDistance(x1, y1, x2, y2):
    return np.sqrt(np.square(x1 - x2) + np.square(y1 - y2))


# Count haversine distance between two points
# /* haversine distance is for earth surface (sphere)
#    lat and lon are in degrees
#    return: approximate distance in kilometers
# */
def haversineDistance(lat1, lon1, lat2, lon2):
    R = 6372.8 # Earth radius in kilometers

    # convert lat and lon difference to radians
    dLat = np.radians(lat2 - lat1)
    dLon = np.radians(lon2 - lon1)

    # convert lat to radians
    lat1 = np.radians(lat1)
    lat2 = np.radians(lat2)

    # haversine formula
```

```
    a = np.sin(dLat/2)**2 + np.sin(dLon/2)**2 * np.cos(lat1) *
np.cos(lat2)
    c = 2 * np.arcsin(np.sqrt(a))

    # calculate final approximate distance in kilometers
    return R * c
```

2.6 gmap.py

```
import gmplot
import places.alunalun as nangor
import os
import webbrowser
import Components as comp
import function as func
import algorithm as algo
import Area as mp

# INITIATE MAP
place = mp.Place(nangor.x, nangor.y, nangor.zoom, nangor.listKoordinat,
nangor.listNodeName, nangor.matriks)
gmap = gmplot.GoogleMapPlotter(place.getCenter()[0], place.getCenter()[1],
place.getZoom())
for i in range(len(place.getListCoordinate())):
    for j in range(len(place.getListCoordinate())):
        if(nangor.matriks[i][j] == 1):
            latitude =
[place.getListCoordinate()[i][0],place.getListCoordinate()[j][0]]
            longitude =
[place.getListCoordinate()[i][1],place.getListCoordinate()[j][1]]
            gmap.scatter(latitude, longitude, 'yellow', size = 7, marker =
False)
            gmap.plot(latitude, longitude, 'blue', edge_width = 3)

for k in range(len(place.getListCoordinate())):
    gmap.text(place.getListCoordinate()[k][0],
place.getListCoordinate()[k][1], place.getListName()[k])

# ALGORITHM
nodeList = func.initiateListNode(nangor.listNodeName, nangor.matriks)
grafMap = comp.Graph(nangor.matriks, nodeList)
```

```
grafMap.convertCoordinatesToWAM(place.getListCoordinate())
algo.heuristicMap(grafMap, 'Patung Tentara Pelajar', place)
# cost, path = algo.uniform_cost_search(grafMap, 'Pungkur', 'Patung
Tentara Pelajar')
cost, path = algo.a_star(grafMap, 'Pungkur', 'Patung Tentara Pelajar')
print(cost)
print(path)

# DRAW PATH
for i in range(len(path)-1):
    x = place.getListName().index(path[i])
    y = place.getListName().index(path[i+1])
    latitude =
[place.getListCoordinate()[x][0],place.getListCoordinate()[y][0]]
    longitude =
[place.getListCoordinate()[x][1],place.getListCoordinate()[y][1]]
    gmap.scatter(latitude, longitude, 'orange', size = 7, marker = True)
    gmap.plot(latitude, longitude, 'red', edge_width = 3)

gmap.draw("mymap.html")

webbrowser.open_new_tab('file://' + os.path.realpath('mymap.html'))
```

2.7 parsing.py

```
from src.core import Components, function

def parse_adjacency_matrix(file_path):
    nodeList = []                          # list of node
    nodeName = []                          # list of string (node name)
    adjacency_matrix = []                  # matrix of int (adjacency matrix)
    with open(file_path, 'r') as f:
        lines = f.readlines()

    # return empty matrix if file is empty
    if len(lines) == 0 or '\n' not in lines:
        return adjacency_matrix, nodeName

    idxName = lines.index('\n')            # find '\n' to seperate matrix and
node name
    if idxName != -1:
```

```
        # convert string to list of list of int (adjacency matrix)
        line = lines[:idxName]
        for l in line:
            adjacency_matrix.append([int(x) for x in l.strip().split()])

        # return empty matrix if matrix is empty or not square
        if len(adjacency_matrix) == 0 or len(adjacency_matrix[0]) == 0 or
len(adjacency_matrix) != len(adjacency_matrix[0]):
            return [], []

        # convert string to list of string (node name)
        nodes = lines[idxName + 1:]
        for node in nodes:
            nodeName = node.strip().split()

        # return empty matrix if node name is empty or not equal to matrix
size
        if len(nodeName) == 0 or len(nodeName) != len(adjacency_matrix):
            return [], []

        # convert list of string to list of node
        nodeList = function.initiateListNode(nodeName, adjacency_matrix)

    # initialize Graph
    graf = Components.Graph(adjacency_matrix, nodeList)
    return graf
```

2.8 gui.py

```
import sys
import webbrowser
import gmplot
import os

import networkx as nx
from PyQt5.QtWidgets import QMainWindow, QApplication, QPushButton,
QFileDialog, QComboBox, QLineEdit, QWidget, QVBoxLayout, QDialog, QLabel,
QPushButton
from PyQt5.QtGui import QFont, QFontMetrics
from PyQt5 import uic
```

```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas

from src.core import algorithm as algo, Area as area, Components as comp,
function as func, parsing as parse
from src.places import itbBdg as gane, itbNangor as nangor, alunalun as
alun, cilacap as cilacap, buahBatu as batu


class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()
        uic.loadUi("./src/gui/gui.ui", self)

        # Initiate attributes
        self.costResult = 0
        self.pathResult = []
        self.nodeNameUI = []
        self.graphUI = None

        #label input
        self.input = self.findChild(QPushButton, "pushButton")
        self.input.clicked.connect(self.pushInputFile)

        #label ucs
        self.ucs = self.findChild(QPushButton, "pushButton_2")
        self.ucs.clicked.connect(self.on_ucs_clicked)
        self.ucs.clicked.connect(self.plot_graph)

        #label a*
        self.a = self.findChild(QPushButton, "pushButton_3")
        self.a.clicked.connect(self.on_a_clicked)
        self.a.clicked.connect(self.plot_graph)

        #dropdown
        self.dropdown = self.findChild(QComboBox, "comboBox_2")
        self.dropdown.addItems(["Input File", "ITB Ganesha", "ITB
Jatinangor", "Alun-Alun Bandung", "Buah Batu", "Cilacap"])
```

```
        self.dropdown.currentIndexChanged.connect(self.dropdownChanged)

        #graph
        self.widget = self.findChild(QWidget, "widget")
        self.widget.setGeometry(800,370,993,645)

        #start
        self.start = self.findChild(QLineEdit, "lineEdit_3")

        #goal
        self.goal = self.findChild(QLineEdit, "lineEdit")

        #show gmap
        self.gmap = self.findChild(QPushButton, "pushButton_4")
        self.gmap.clicked.connect(self.showGmap)

        # cost
        self.cost = self.findChild(QLabel, "label_10")

        # showmap
        self.map = self.findChild(QPushButton, "pushButton_5")
        self.map.clicked.connect(self.showmap)
        self.widget2 = QWidget(self)

        # path
        self.path = self.findChild(QLabel, "label_11")

        self.show()

    def showErrorMessage(self, message):
        dialog = QDialog(self)
        dialog.setWindowTitle("Error")
        layout = QVBoxLayout(dialog)
        layout.addWidget(QLabel(message))
        button = QPushButton("OK", dialog)
        button.clicked.connect(dialog.accept)
        layout.addWidget(button)
        dialog.exec_()

    # input file
```

```
    def pushInputFile(self):
        self.nodeNameUI = []
        try :
            inputFile = QFileDialog.getOpenFileName(self, 'Open file',
'c:\\', "Text files (*.txt)")
            if inputFile:
                self.graphUI = parse.parse_adjacency_matrix(inputFile[0])
                for i in range(len(self.graphUI.getListNode())):
                    self.nodeNameUI.append(self.graphUI.getNameNode(i))
        except :
            self.showErrorMessage("Please input file first")

    # input start
    def input(self):
        global input
        input = self.start.text()

    # input goal
    def goal(self):
        global goal
        goal = self.goal.text()

    #plot graph
    def plot_graph(self):
        Graph = nx.Graph()
        for i in Graph.nodes():
            Graph.nodes[i]['label'] = self.nodeNameUI[i]
        for i in range(len(self.nodeNameUI)):
            for j in range(i+1, len(self.nodeNameUI)):
                if self.graphUI.getMatrix()[i][j] != 0:
                    Graph.add_edge(self.nodeNameUI[i], self.nodeNameUI[j],
weight=self.graphUI.getMatrix()[i][j])
        fig = Figure(figsize=(10,5.3), dpi=100)
        canvas = FigureCanvas(fig)
        pos = nx.spring_layout(Graph)
        ax = fig.add_subplot(111)
        edge_colors = ['red' if (u, v) in zip(self.pathResult,
self.pathResult[1:]) else 'black' for u, v in Graph.edges()]
        nx.draw_networkx_edge_labels(Graph, pos,
edge_labels=nx.get_edge_attributes(Graph, 'weight'))
```

```
        nx.draw(Graph, pos, with_labels=True, node_size=300,
node_color='green', edge_color=edge_colors, font_size=10, ax=ax)
        plt.axis('off')
        canvas.setParent(self.widget)
        canvas.show()

    def initiateMapToGraph(self, place: area.Area):
        self.nodeNameUI = []
        nodeList = func.initiateListNode(place.getListName(),
place.getMatrix())
        self.graphUI = comp.Graph(place.getMatrix(), nodeList)
        self.graphUI.convertCoordinatesToWAM(place.getListCoordinate())
        for i in range(len(self.graphUI.getListNode())):
            self.nodeNameUI.append(self.graphUI.getNameNode(i))

    #dropdown
    def dropdownChanged(self):
        if self.dropdown.currentText() == "Input File":
            pass
        else:
            global place

            if self.dropdown.currentText() == "ITB Ganesha":
                place = area.Area(gane.x, gane.y, gane.zoom,
gane.listKoordinat, gane.listNodeName, gane.matriks)

            elif self.dropdown.currentText() == "ITB Jatinangor":
                place = area.Area(nangor.x, nangor.y, nangor.zoom,
nangor.listKoordinat, nangor.listNodeName, nangor.matriks)

            elif self.dropdown.currentText() == "Alun-Alun Bandung":
                place = area.Area(alun.x, alun.y, alun.zoom,
alun.listKoordinat, alun.listNodeName, alun.matriks)

            elif self.dropdown.currentText() == "Buah Batu":
                place = area.Area(batu.x, batu.y, batu.zoom,
batu.listKoordinat, batu.listNodeName, batu.matriks)

            elif self.dropdown.currentText() == "Cilacap":
```

```
                place = area.Area(cilacap.x, cilacap.y, cilacap.zoom,
cilacap.listKoordinat, cilacap.listNodeName, cilacap.matriks)

            self.initiateMapToGraph(place)

    #show map
    def showmap(self):
        if self.dropdown.currentText() == "ITB Ganesha":
            self.widget2 = QWidget()
            self.widget2.resize(800, 600)
            label = QLabel(self.widget2)
            Gane = "ITB Ganesha\n 1. Gerbang Utama\n 2. Gedung Sipil\n 3.
Gedung CIBE\n 4. CC Barat\n 5. CC Timur\n 6. Lapangan Cinta\n 7. Gedung
SAPPK\n 8. Gedung SR \n 9. Sekretariat IMG\n 10. Gedung Lingkungan\n 11.
GKU Barat\n 12. Labtek VII\n 13. Labtek III\n 14. Gedung Kimia\n 15.
Gedung FTTM\n 16. Gedung FITB\n 17. Gedung CAS\n 18. Gedung CRCS\n 19.
Perpustakaan\n 20. Gedung SBM\n 21. Sunken Court\n 22. Pusat Gema\n 23.
Plaza Widya\n 24. Aula Barat-Timur"
            label.setText(str(Gane))
            self.widget2.setWindowTitle("ITB Ganesha")
            self.widget2.show()
        elif self.dropdown.currentText() == "ITB Jatinangor":
            self.widget2 = QWidget()
            self.widget2.resize(500, 300)
            label = QLabel(self.widget2)
            Nangor = "ITB Jatinangor\n 1. Gerbang Utama\n 2. Bundaran\n 3.
GKU\n 4. Water Treatment Plan\n 5. Asrama Mahasiswa\n 6. GSG\n 7. Koica\n
8. Rektorat\n 9. GKU 1\n 10. Asrama Dosen"
            label.setText(str(Nangor))
            self.widget2.setWindowTitle("ITB Jatinangor")
            self.widget2.show()
        elif self.dropdown.currentText() == "Alun-Alun Bandung":
            self.widget2 = QWidget()
            self.widget2.resize(800, 600)
            label = QLabel(self.widget2)
            Alun = "Alun-Alun Bandung\n 1. Paskal\n 2. Parapan Kebon
Jati\n 3. Sudirman\n 4. Stasiun Bandung\n 5. Cibadak\n 6. Astana Anyar\n
7. Pasir Koja\n 8. Norsefiicden\n 9. Otto Iskandar Dinata\n 10. Pungkur\n
11. Patung Tentara Pelajar\n 12. Braga Citywalk\n 13. Braga\n 14. Air
Mancur Asia Afrika\n 15. Jalan ABC\n 16. Asia Afrika\n 17. Mural Asia
```

```
Afrika\n 18. Jalan Homan\n 19. Alun-Alun Bandung\n 20. Pendopo Kota
Bandung\n 21. Pasar Baru"
            label.setText(str(Alun))
            self.widget2.setWindowTitle("Alun-Alun Bandung")
            self.widget2.show()
        elif self.dropdown.currentText() == "Buah Batu":
            self. widget2 = QWidget()
            self.widget2.resize(800, 600)
            label = QLabel(self.widget2)
            Batu = "Buah batu\n 1. Jalan Neptunus Barat II - III\n 2.
Jalan Neptunus Barat III - V\n 3. Jalan Neptunus Barat IV - Neptunus Raya
Barat\n 4. Jalan Neptunus Barat VI - Neptunus Raya Barat\n 5. Jalan
Neptunus Barat III - Neptunus Raya Barat\n 6. Jalan Neptunus Barat II -
Neptunus Raya Barat - Neptunus Tengah\n 6. Jalan Neptunus Barat I -
Neptunus Barat - Neptunus Raya\n 7. Jalan Neptunus Tengah II - Neptunus
Tengah"
            label.setText(str(Batu))
            self.widget2.setWindowTitle("Buah Batu")
            self.widget2.show()
        elif self.dropdown.currentText() == "Cilacap":
            self.widget2 = QWidget()
            self.widget2.resize(500, 300)
            label = QLabel(self.widget2)
            Cilacap = "Cilacap\n 1. Pelabuhan Penyebrangan Sleko\n 2.
Pelabuhan Internasional Tanjung Intan\n 3. Cilacap Town Square\n 4.
Stasiun Cilacap\n 5. Pendopo Wijaya Kusuma Sakti\n 6. Masjid Agung
Darussalam\n 7. Soedirman Soccer Field\n 8. Tugu Nelayan\n 9. Klenteng Lam
Tjeng Kiong"
            label.setText(str(Cilacap))
            self.widget2.setWindowTitle("Cilacap")
            self.widget2.show()
        elif self.dropdown.currentText() == "Input File":
            self.widget2 = QWidget()
            self.widget2.resize(100,100)
            label = QLabel(self.widget2)
            label.setText("You Choose Input File")
            self.widget2.setWindowTitle("Input File")
            self.widget2.show()

            self.initiateMapToGraph(place)
```

```
    #ucs
    def chooseUCS(self):
        cost = 0
        path = []
        try:
            cost, path = algo.uniform_cost_search(self.graphUI,
self.start.text(), self.goal.text())

            font = QFont()
            font.setPointSize(10)
            self.cost.setFont(font)
            self.cost.setText(str(cost))
            path_font = QFont()
            path_font.setPointSize(8)
            self.path.setFont(font)
            self.path.setText(str(path))
            self.path.setAdjustSize(True)

        except:
            if self.start.text() == "":
                self.showErrorMessage("Start is empty")

            elif self.start.text() not in self.nodeNameUI:
                self.showErrorMessage("Start is not in the node")

            if self.goal.text() == "":
                self.showErrorMessage("Goal is empty")

            elif self.goal.text() not in self.nodeNameUI:
                self.showErrorMessage("Goal is not in the node")

        return cost, path


    #astar
    def chooseA(self):
        cost = 0
        path = []
        try:
```

```
            if self.dropdown.currentText() == "Input File":
                algo.heuristic(self.graphUI, self.goal.text())
            else:
                algo.heuristicMap(self.graphUI, self.goal.text(), place)

            cost, path = algo.a_star(self.graphUI, self.start.text(),
self.goal.text())

            font = QFont()
            font.setPointSize(10)
            self.cost.setFont(font)
            self.cost.setText(str(cost))
            self.path.setFont(font)
            self.path.setText(str(path))
            self.path.setAdjustSize(True)

        except:
            if self.start.text() == "":
                self.showErrorMessage("Start is empty")

            elif self.start.text() not in self.nodeNameUI:
                self.showErrorMessage("Start is not in the node")

            if self.goal.text() == "":
                self.showErrorMessage("Goal is empty")

            elif self.goal.text() not in self.nodeNameUI:
                self.showErrorMessage("Goal is not in the node")

        return cost, path

    # Assign costResult and pathResult on click event
    def on_ucs_clicked(self):
        self.costResult, self.pathResult = self.chooseUCS()

    def on_a_clicked(self):
        self.costResult, self.pathResult = self.chooseA()

    # show gmap
    def showGmap(self):
```

```
        try:
            gmap = gmplot.GoogleMapPlotter(place.getCenter()[0],
place.getCenter()[1], place.getZoom())
            for i in range(len(self.graphUI.getMatrix())):
                for j in range(len(self.graphUI.getMatrix())):
                    if(self.graphUI.getMatrix()[i][j] > 0):
                        latitude =
[place.getListCoordinate()[i][0],place.getListCoordinate()[j][0]]
                        longitude =
[place.getListCoordinate()[i][1],place.getListCoordinate()[j][1]]
                        gmap.scatter(latitude, longitude, 'yellow', size =
7, marker = False)
                        gmap.plot(latitude, longitude, 'blue', edge_width
= 3)

            for k in range(len(place.getMatrix())):
                gmap.text(place.getListCoordinate()[k][0],
place.getListCoordinate()[k][1], place.getListName()[k])

            for i in range(len(self.pathResult)-1):
                x = place.getListName().index(self.pathResult[i])
                y = place.getListName().index(self.pathResult[i+1])
                latitude =
[place.getListCoordinate()[x][0],place.getListCoordinate()[y][0]]
                longitude =
[place.getListCoordinate()[x][1],place.getListCoordinate()[y][1]]
                gmap.scatter(latitude, longitude, 'orange', size = 7,
marker = True)
                gmap.plot(latitude, longitude, 'red', edge_width = 3)

            gmap.draw("./mymap.html")

            webbrowser.open_new_tab('file://' +
os.path.realpath('mymap.html'))
        except:
            self.showErrorMessage("Please choose the algorithm")

def initiateUI():
    app = QApplication(sys.argv)
    window = UI()
```

```
    sys.exit(app.exec_())

if __name__ == '__main__':
    # import sys
    app = QApplication(sys.argv)
    window = UI()
    sys.exit(app.exec_())
```

2.9 alunalun.py

```
x = -6.917921152550039
y = 107.60374055640584
zoom = 17

listKoordinat = [(-6.914763482418373, 107.59806082445583),      # Paskal
[0]
                 (-6.916314446905801, 107.59819713067857),      # Parapan
Kebon Jati [1]
                 (-6.920102574626576, 107.59838782222505),      # Sudirman
[2]
                 (-6.915782319451493, 107.60449944984325),      # Stasiun
Bandung [3]
                 (-6.92139789132258, 107.5983944107498),        # Cibadak
[4]
                 (-6.9268349273287075, 107.59985715600061),     # Astana
Anyar [5]
                 (-6.927243471429872, 107.60362566069243),      # Pasir
Koja [6]
                 (-6.920815316114, 107.6041037562899),          #
Norsefiicden [7]
                 (-6.92208984114456, 107.60400910131901),       # Otto
Iskandar Dinata [8]
                 (-6.927390546790986, 107.60592904774549),     # Pungkur
[9]
                 (-6.915389760946487, 107.607441182319 69),     # Patung
Tentara Pelajar [10]
                 (-6.916239625489668, 107.60891830680069),      # Braga
Citywalk [11]
```

```
                (-6.919708211715332, 107.609905698286),        # Braga
[12]
                (-6.919735443811863, 107.60889424224001),      # Air
Mancur Asia Afrika [13]
                (-6.918880823058204, 107.6067069791089),       # Jalan
ABC [14]
                (-6.921441946335599, 107.60979549903513),      # Asia
Afrika [15]
                (-6.92131144369633, 107.60874097849286),       # Mural
Asia Afrika [16]
                (-6.922782395509796, 107.60977084474239),      # Jalan
Homan [17]
                (-6.921225265358179, 107.60762122080159),      #
Alun-Alun Bandung [18]
                (-6.922512286325299, 107.60742554184935),      # Pendopo
Kota Bandung [19]
                (-6.918295574544841, 107.60431298317197)]      # Pasar
Baru [20]

listNodeName = ["Paskal",                           # [0] x
                "Parapan Kebon Jati",               # [1] x
                "Sudirman",                         # [2] x
                "Stasiun Bandung",                  # [3] x
                "Cibadak",                          # [4] x
                "Astana Anyar",                     # [5] x
                "Pasir Koja",                       # [6] x
                "Norsefiicden",                     # [7] x
                "Otto Iskandar Dinata",             # [8] x
                "Pungkur",                          # [9] x
                "Patung Tentara Pelajar",           # [10] x
                "Braga Citywalk",                   # [11] x
                "Braga",                            # [12] x
                "Air Mancur Asia Afrika",           # [13] x
                "Jalan ABC",                        # [14] x
                "Asia Afrika",                      # [15] x
                "Mural Asia Afrika",                # [16] x
                "Jalan Homan",                      # [17] x
                "Alun-Alun Bandung",                # [18] x
                "Pendopo Kota Bandung",             # [19] x
                "Pasar Baru"]                       # [20] x
```

```
#            0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
matriks = [ [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1],
            [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1],
            [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0],
            [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0],
            [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1],    # [14]
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0],
            [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0],
```

```
            [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0],
            [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0]
           ]
```

2.10 itbBdg.py

```
x = -6.89132022515749
y = 107.61061618283757
zoom = 17

listKoordinat = [(-6.8931357321569795, 107.61043417848474),      # Gerbang
Utama [0]
                 (-6.8926481246340465, 107.60876860281701),      # Gedung
Sipil [1]
                 (-6.891045439979195, 107.60868333225132),       # Gedung
CIBE [2]
                 (-6.89102682287638, 107.60971545320055),        # CC Barat
[3]
                 (-6.891019098250985, 107.6110472483217),        # CC Timur
[4]
                 (-6.891911273407023, 107.61039174777933),       # Lapangan
Cinta [5]
                 (-6.891357415665471, 107.61104060119305),       # Gedung
SAPPK [6]
                 (-6.892441432391727, 107.61184871738482),       # Gedung
SR [7]
                 (-6.891324641487534, 107.6121771135574),        #
Sekretariat IMG  [8]
                 (-6.8909780544008346, 107.61154244628963),      # Gedung
Lingkungan [9]
                 (-6.89013568927372, 107.60904936987745),        # GKU
Barat [10]
                 (-6.889858493408902, 107.61152535852719),       # Labtek
VII [11]
                 (-6.888703707464329, 107.60912116012126),       # Labtek
III [12]
                 (-6.889445935319975, 107.61153615159377),       # Gedung
Kimia [13]
```

```
                (-6.889100770793402, 107.61153640188064),       # Gedung
FTTM [14]
                (-6.888696387683714, 107.61151220623402),       # Gedung
FITB [15]
                (-6.8881713215082385, 107.61148441227157),      # Gedung
CAS [16]
                (-6.887865079092212, 107.61144904859228),       # Gedung
CRCS [17]
                (-6.887843022137563, 107.610493157145480),      #
Perpustakaan [18]
                (-6.887996245289205, 107.60920683118938),       # Gedung
SBM [19]
                (-6.888680340973249, 107.61035240783394),       # Sunken
Court [20]
                (-6.88989395284103, 107.61035124543669),        # Pusat
Gema [21]
                (-6.891022084739352, 107.6103473815715),        # Plaza
Widya [22]
                (-6.892600770578635, 107.61042090218989)]       # Aula
Barat-Timur [23]

listNodeName = ['Gerbang Utama',          # 0 x
                'Gedung Sipil',           # 1 x
                'Gedung CIBE',            # 2 x
                'CC Barat',               # 3 x
                'CC Timur',               # 4 x
                'Lapangan Cinta',         # 5 x
                'Gedung SAPPK',           # 6 x
                'Gedung SR',              # 7 x
                'Sekretariat IMG',        # 8 x
                'Gedung Lingkungan',      # 9 x
                'GKU Barat',              # 10 x
                'Labtek VII',             # 11 x
                'Labtek III',             # 12 x
                'Gedung Kimia',           # 13 x
                'Gedung FTTM',            # 14 x
                'Gedung FITB',            # 15 x
                'Gedung CAS',             # 16 x
                'Gedung CRCS',            # 17 x
                'Perpustakaan',           # 18 x
```

```
                'Gedung SBM',           # 19 x
                'Sunken Court',         # 20 x
                'Pusat Gema',           # 21 x
                'Plaza Widya',          # 22
                'Aula Barat-Timur']     # 23

#            0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 1, 2, 3
matriks = [ [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1],
            [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0],
            [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0],
            [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1],
            [0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1],
            [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 0],
```

```
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
1, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
0, 1, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 0],
            [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0],
            [1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0] ]
```

2.11 itbNangor.py

```
x = -6.929913757931639
y = 107.7694522966798
zoom = 17

listKoordinat = [(-6.933658447500457, 107.76837589259873),      # Gerbang
Utama [0]
                 (-6.931800546112187, 107.76887045663597),      # Bundaran
[1]
                 (-6.930375582337828, 107.76852616194915),      # GKU [2]
                 (-6.92821930118407, 107.7675663263493),        # Water
Treatment Plan [3]
                 (-6.926687358593196, 107.76774315232038),      # Asrama
Mahasiswa [4]
                 (-6.926328308837532, 107.76914168508372),      # GSG [5]
                 (-6.927596950084163, 107.77018656588393),      # Koica
[6]
                 (-6.928259195416854, 107.77084564454252),      # Rektorat
[7]
                 (-6.929058076076697, 107.77002881540378),      # GKU 1
[8]
                 (-6.93263442078499, 107.77011505517082)]       # Asrama
Dosen [9]
```

```
listNodeName = ["Gerbang Utama",
                "Bundaran",
                "GKU",
                "Water Treatment Plan",
                "Asrama Mahasiswa",
                "GSG",
                "Koica",
                "Rektorat",
                "GKU 1",
                "Asrama Dosen"]

matriks = [ [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 1, 0, 0, 0, 0, 0, 0, 1],
            [0, 1, 0, 1, 0, 0, 0, 0, 1, 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, 1, 0, 0, 0],
            [0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
            [0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
            [0, 0, 1, 0, 0, 0, 0, 1, 0, 1],
            [0, 1, 0, 0, 0, 0, 0, 0, 1, 0]]
```

2.12 main.py

```
import src

if __name__ == "__main__":
    src.gui.initiateUI()
```

2.13 cilacap.py

```
x = -7.72395166976066
y = 109.01107447074276
zoom = 17

listKoordinat = [(-7.727070648993984, 108.99678928824123),
                 (-7.7323503784489285, 108.99288251171176),
                 (-7.726811094404932, 109.00957021107152),
                 (-7.735726751844093, 109.00711636242185),
                 (-7.725842721001097, 109.00968318727703),
                 (-7.727282119350269, 109.0086545888975),
```

```
                (-7.72676251061673, 109.01208976773698),
                (-7.727780505868129, 109.00951182685097),
                (-7.732823850146193, 109.0091025987741)]


listNodeName = ["Pelabuhan Penyebrangan Sleko",
                "Pelabuhan Internasional Tanjung Intan",
                "Cilacap Town Square",
                "Stasiun Cilacap",
                "Pendopo Wijaya Kusuma Sakti",
                "Masjid Agung Darussalam",
                "Soedirman Soccer Field",
                "Tugu Nelayan",
                "Klenteng Lam Tjeng Kiong"]

#            1, 2, 3, 4, 5, 6, 7, 8, 9
matriks = [ [0, 1, 0, 0, 0, 1, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0, 1],
            [0, 0, 0, 0, 1, 1, 0, 1, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 1],
            [0, 0, 1, 0, 0, 1, 0, 1, 0],
            [1, 0, 1, 0, 1, 0, 0, 1, 0],
            [0, 0, 0, 0, 0, 0, 0, 1, 0],
            [0, 0, 1, 0, 1, 1, 1, 0, 1],
            [0, 1, 0, 1, 0, 0, 0, 1, 0]]
```

2.14 buahBatu.py

```
x = -6.948021406199452
y = 107.65561722740172
zoom = 18


listKoordinat = [   [-6.948021406199452, 107.65561722740172],
                    [-6.9476480695358065, 107.65565807991213],
                    [-6.94753142760766, 107.65535394662533],
                    [-6.947221557218974, 107.6553960359341],
                    [-6.947144960012765, 107.65540831198251],
                    [-6.948100683361098, 107.65590636884221],
                    [-6.948402149724846, 107.65584082025177],
                    [-6.948209037854483, 107.65700094013205] ]
```

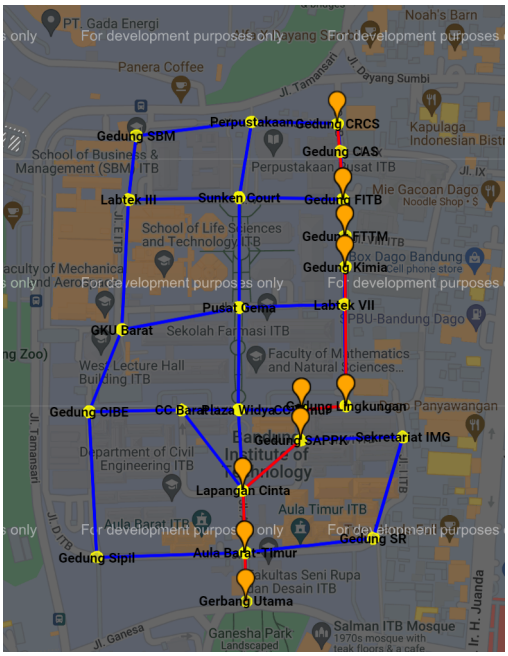
```
listNodeName = [ 'Jalan Neptunus Barat II - III',
                 'Jalan Neptunus Barat III - IV',
                 'Jalan Neptunus Barat IV - Neptunus Raya Barat',
                 'Jalan Neptunus Barat VI - Neptunus Raya Barat',
                 'Jalan Neptunus Barat III - Neptunus Raya Barat',
                 'Jalan Neptunus Barat II - Neptunus Barat - Neptunus
Tengah',
                 'Jalan Neptunus Barat I - Neptunus Barat - Neptunus Raya',
                 'Jalan Neptunus Tengah II - Neptunus Tengah' ]

matriks = [ [0, 1, 0, 0, 0, 1, 0, 0],
            [1, 0, 1, 0, 1, 0, 0, 0],
            [0, 1, 0, 1, 0, 0, 0, 0],
            [0, 0, 1, 0, 1, 0, 0, 0],
            [0, 1, 0, 1, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 1, 1],
            [0, 0, 0, 0, 0, 1, 0, 0],
            [0, 0, 0, 0, 0, 1, 0, 0] ]
```

BAB III
HASIL PENGUJIAN

3.1 Peta Jalan Sekitar Kampus ITB Ganesha

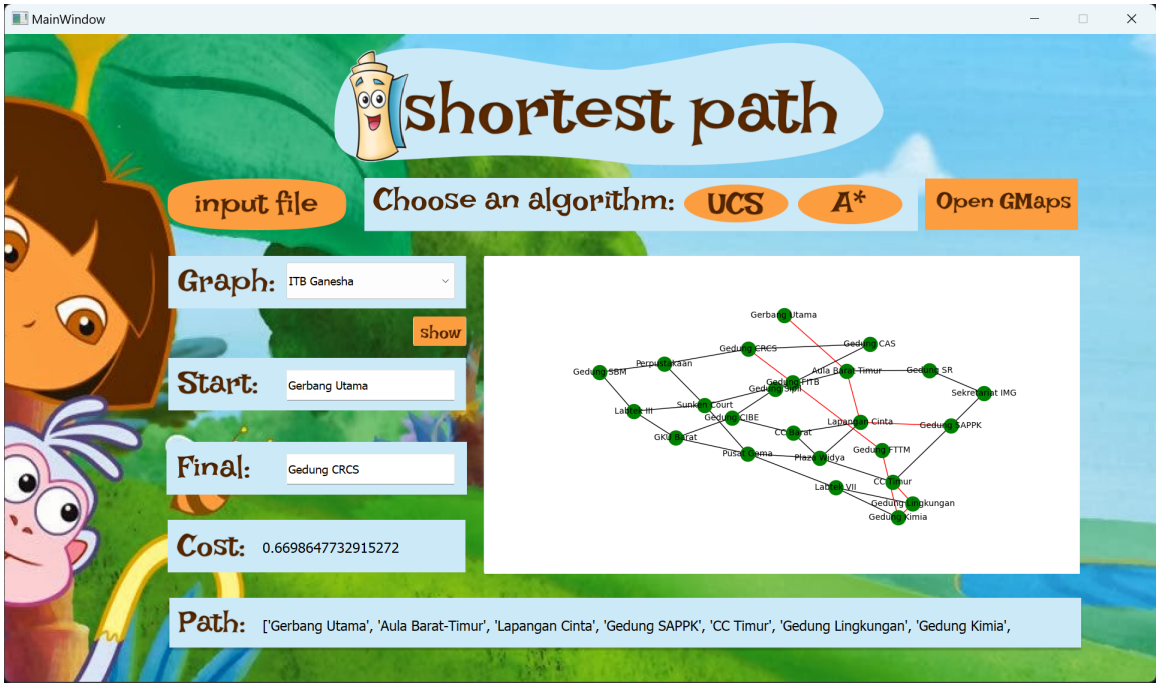
3.1.1 Pengujian A* dari Gerbang Utama ke Gedung CRCS

Gambar 3.1.1.1 Hasil Pengujian A* dari Gerbang Utama ke Gedung CRCS

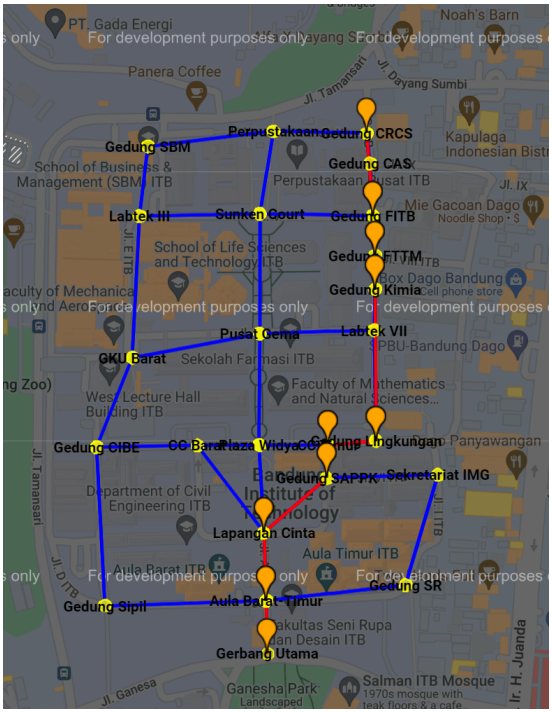


Gambar 3.1.1.2 Plot Peta pada Google Map

3.1.2 Pengujian UCS dari Gerbang Utama ke Gedung CRCS



Gambar 3.1.2.1 Hasil Pengujian UCS dari Gerbang Utama ke Gedung CRCS

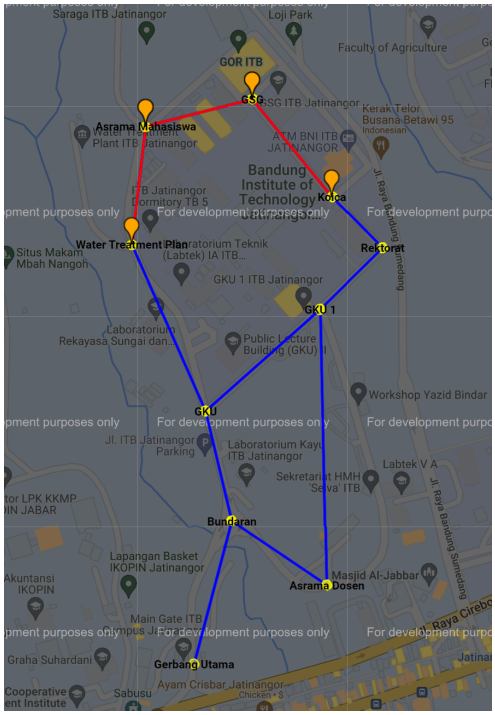


Gambar 3.1.2.2 Plot Peta pada Google Map

3.2 Peta Jalan Sekitar Kampus ITB Jatinangor

3.2.1 Pengujian A* dari Water Treatment Plan ke Koica

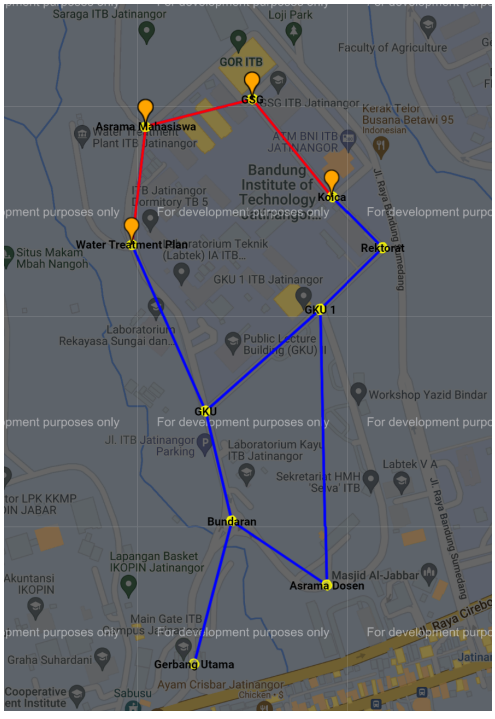
Gambar 3.2.1.1 Hasil Pengujian A* dari Water Treatment Plan ke Gedung Koica



Gambar 3.2.1.2 Plot Peta pada Google Map

3.2.2 Pengujian UCS dari Water Treatment Plan ke Koica

Gambar 3.2.2.1 Hasil Pengujian UCS dari Water Treatment Plan ke Gedung Koica

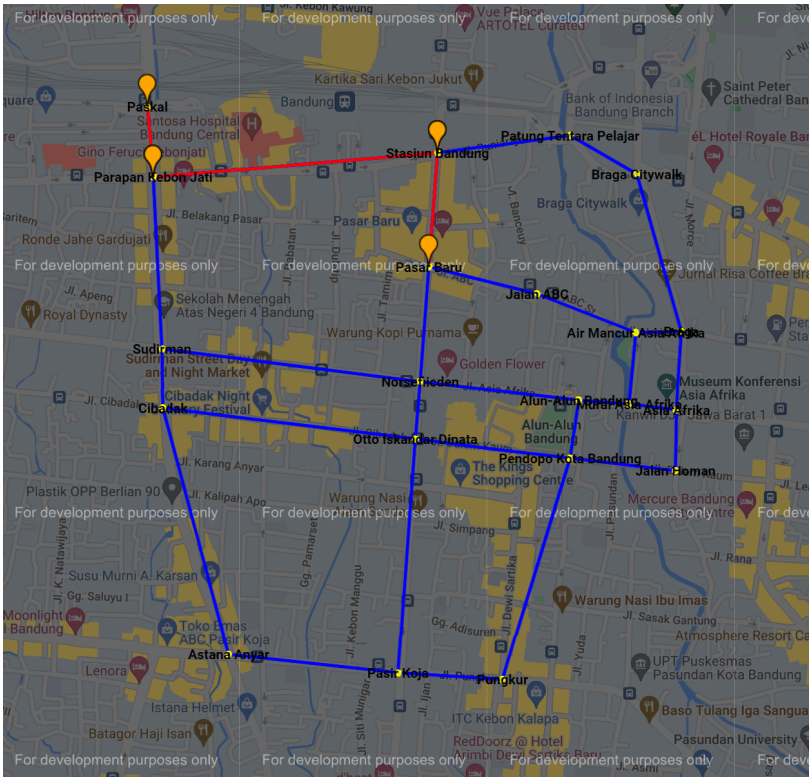


Gambar 3.2.2.2 Plot Peta pada Google Map

3.3 Peta Sekitar Alun-Alun Bandung

3.3.1 Pengujian A* dari Paskal ke Pasar Baru

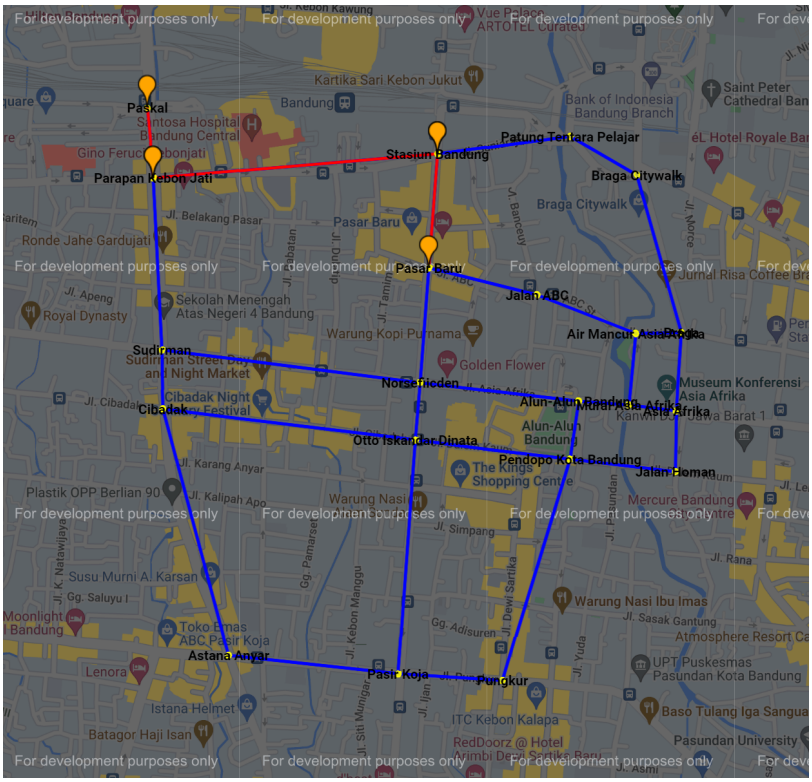
Gambar 3.3.1.1 Hasil Pengujian A* dari Paskal ke Pasar Baru



Gambar 3.3.1.2 Plot Peta pada Google Map

3.3.2 Pengujian UCS dari Paskal ke Pasar Baru

Gambar 3.3.2.1 Hasil Pengujian UCS dari Paskal ke Pasar Baru

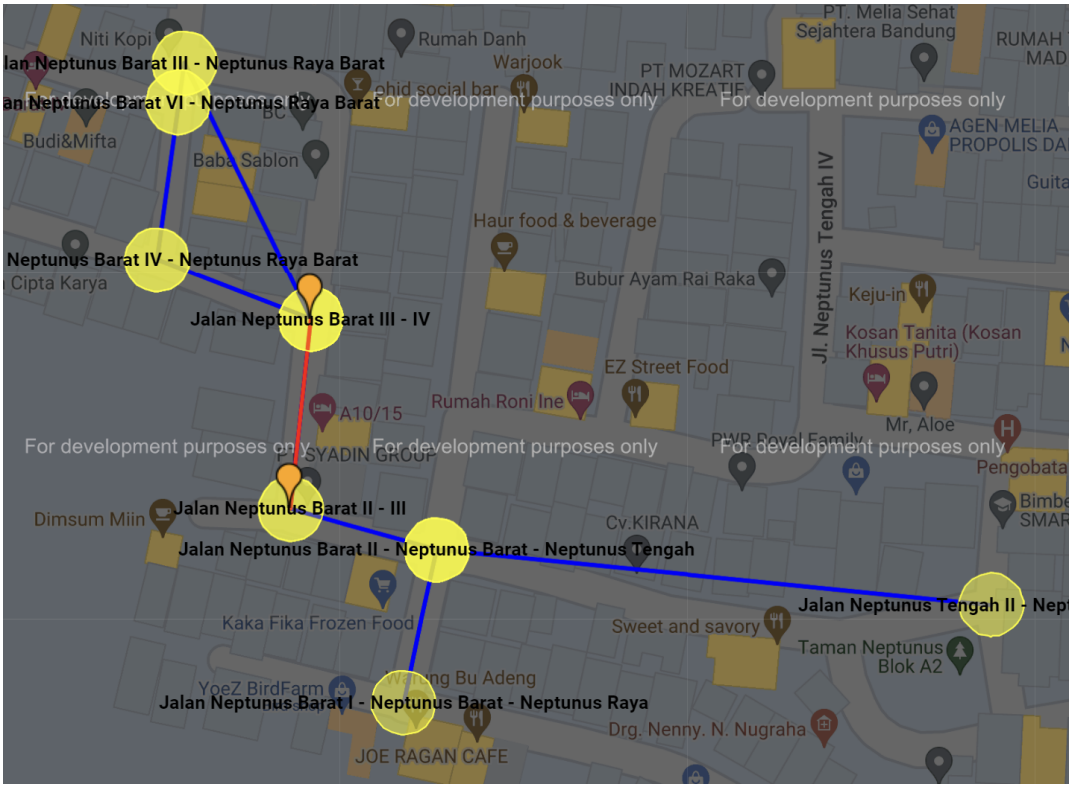


Gambar 3.3.2.2 Plot Peta pada Google Map

3.4 Peta Sekitar Buah Batu

3.4.1 Pengujian A* dari Jalan Neptunus Barat II - II ke Jalan Neptunus Barat III - IV

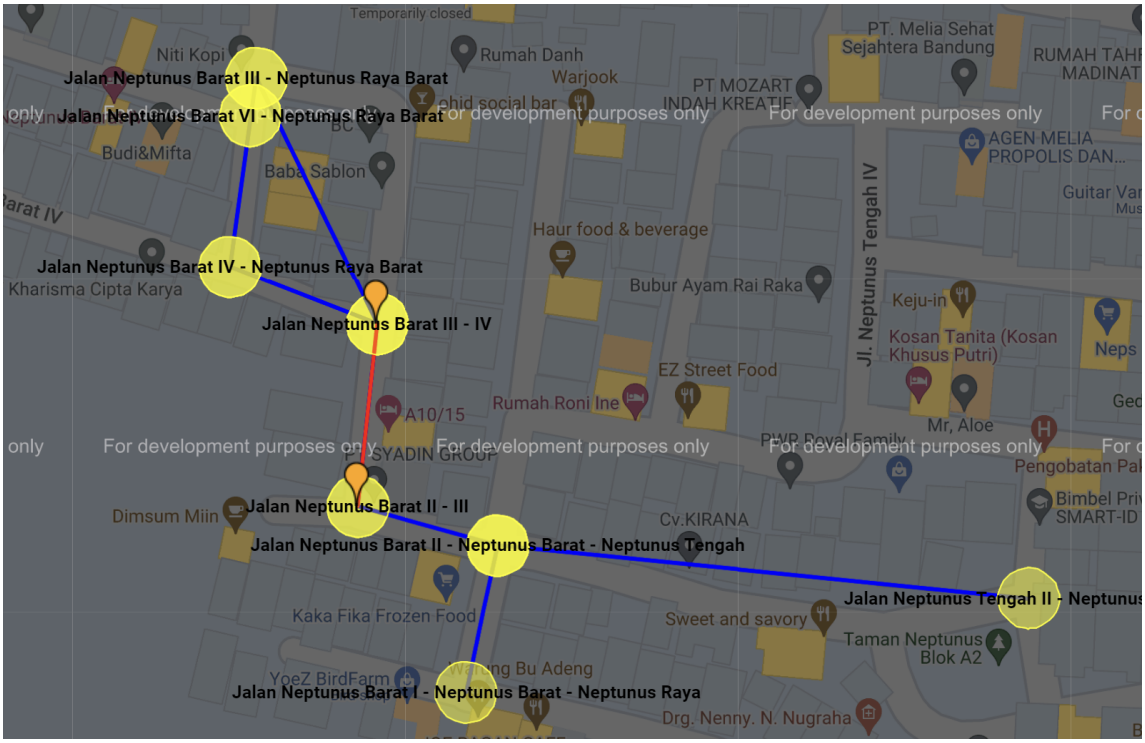
Gambar 3.4.1.1 Hasil Pengujian A* dari Jalan Neptunus Barat II - II ke Jalan Neptunus Barat III - IV



Gambar 3.4.2.2 Plot Peta pada Google Map

3.4.2 Pengujian UCS dari Jalan Neptunus Barat II - II ke Jalan Neptunus Barat III - IV

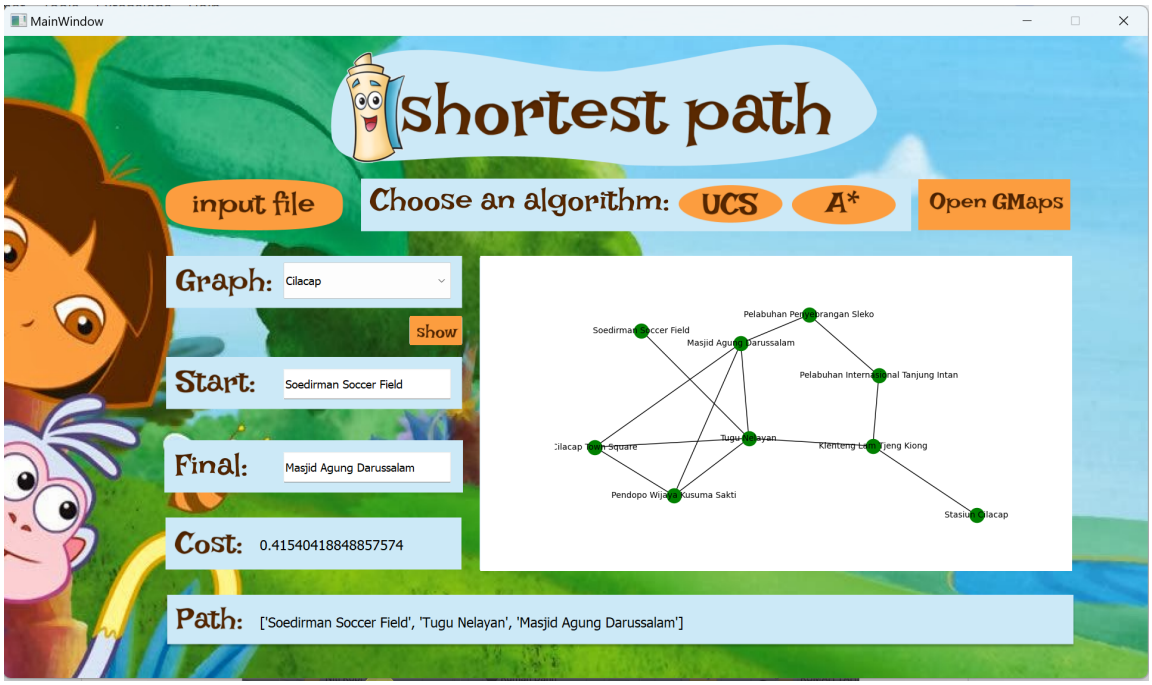
Gambar 3.4.2.2 Hasil Pengujian UCS dari Jalan Neptunus Barat II - II ke Jalan Neptunus Barat III - IV



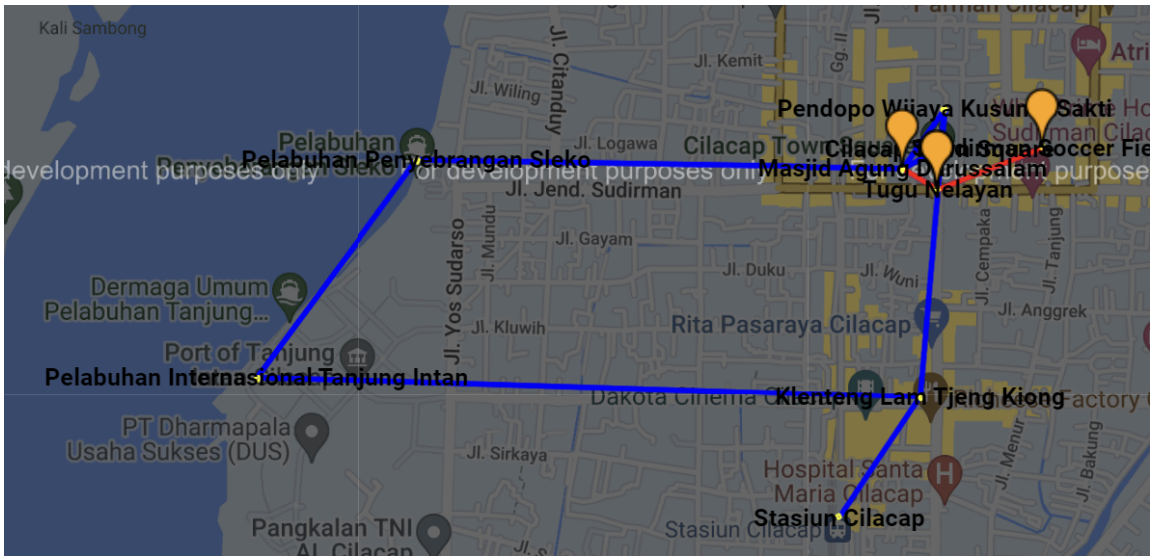
Gambar 3.4.2.2 Plot Peta pada Google Map

3.5 Peta Sekitar Kota Asal

3.5.1 Pengujian A* dari Soedirman Soccer Field ke Masjid Agung Darussalam



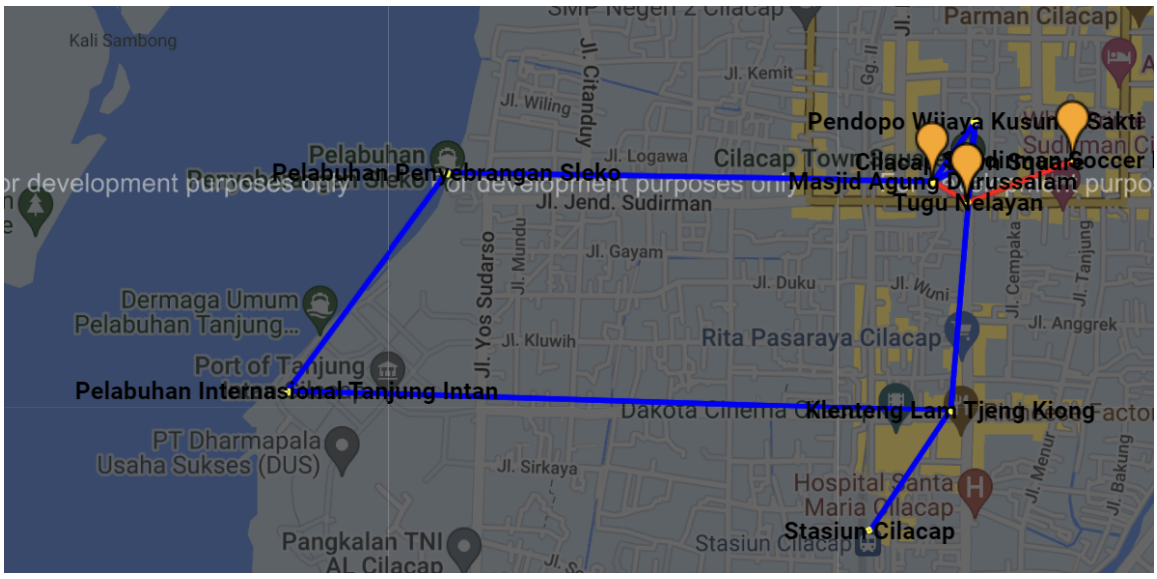
Gambar 3.5.1.1 Hasil Pengujian A* dari Soedirman Soccer Field ke Masjid Agung Darussalam



Gambar 3.5.1.2 Plot Peta pada Google Map

3.5.2 Pengujian UCS dari Soedirman Soccer Field ke Masjid Agung Darussalam

Gambar 3.5.2.1 Hasil Pengujian UCS dari Soedirman Soccer Field ke Masjid Agung Darussalam



Gambar 3.5.2.2 Plot Peta pada Google Map

BAB IV
LAMPIRAN

4.1 Tautan Repository

https://github.com/goodgirlwannabe/Tucil3_13521006_13521023

4.2 Tabel Penilaian

No	Poin	Keterangan
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	- Menamiplkan peta serta lintasan terpendek pada peta ✔

4.3 Kesimpulan

Algoritma *Uniform-Cost Search* dan A* merupakan algoritma yang dapat dimanfaatkan dalam kehidupan sehari-hari, khususnya dalam permasalahan pencarian rute terpendek. UCS bekerja dengan menyisipkan node sumber lalu memasukkan satu persatu bila diperlukan. Sedangkan A* bekerja dengan memperhitungkan fungsi heuristik. Pada tugas besar 2 IF2211 Strategi Algoritma ini, penulis telah mengimplementasikan kedua algoritma tersebut dalam program Shortest Path.

4.4 Komentar

Komentar penulis dalam mengerjakan Tugas Kecil IF2211 Strategi Algoritma adalah sebagai berikut.

13521006	Seru tapi ribet.
13521023	Sebaiknya spesifikasi bisa diperjelas atau dikurangi kalimat-kalimat yang ambigu