LAPORAN

TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Oleh:

Hidayatullah Wildan Ghaly Buchary (13521015)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG BANDUNG

2022/2023

DAFTAR ISI

BAB I

PENDAHULUAN

Pada era teknologi yang semakin maju seperti sekarang ini, banyak sekali aplikasi yang menggunakan algoritma untuk membantu pengguna dalam mengambil keputusan, salah satunya adalah algoritma pencarian rute terpendek. Algoritma ini banyak digunakan dalam navigasi untuk menentukan rute tercepat antara dua titik yang diberikan.

Dalam tugas ini, penulis akan membahas implementasi dua algoritma pencarian rute terpendek yaitu Uniform Cost Search (UCS) dan A* untuk menentukan lintasan terpendek. Algoritma UCS merupakan algoritma pencarian yang menggunakan strategi best-first search dengan mempertimbangkan biaya yang terkait dengan setiap tindakan yang diambil dalam pencarian. Sedangkan algoritma A* merupakan pengembangan dari algoritma UCS yang menambahkan heuristik sebagai fungsi penilaian yang lebih kompleks.

Penentuan lintasan terpendek sangat penting dalam aplikasi navigasi, khususnya dalam situasi yang membutuhkan kecepatan dan ketepatan dalam menentukan rute. Dalam laporan ini, penulis akan menjelaskan implementasi kedua algoritma serta melakukan perbandingan performa antara keduanya dalam menyelesaikan permasalahan penentuan lintasan terpendek.

BAB II
DESKRIPSI TUGAS

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.

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.

Bonus: Bonus nilai diberikan jika dapat menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta (berupa jalan yang diberi warna). Simpul graf diperoleh dari peta (menggunakan API Google Map) dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Euclidean.

Peta jalan yang digunakan sebagai kasus uji adalah:

1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara
2. Peta jalan sekitar Alun-alun Bandung
3. Peta jalan sekitar Buahbatu atau Bandung Selatan
4. Peta jalan sebuah kawasan di kota asalmu

BAB III

LANGKAH JALANNYA PROGRAM

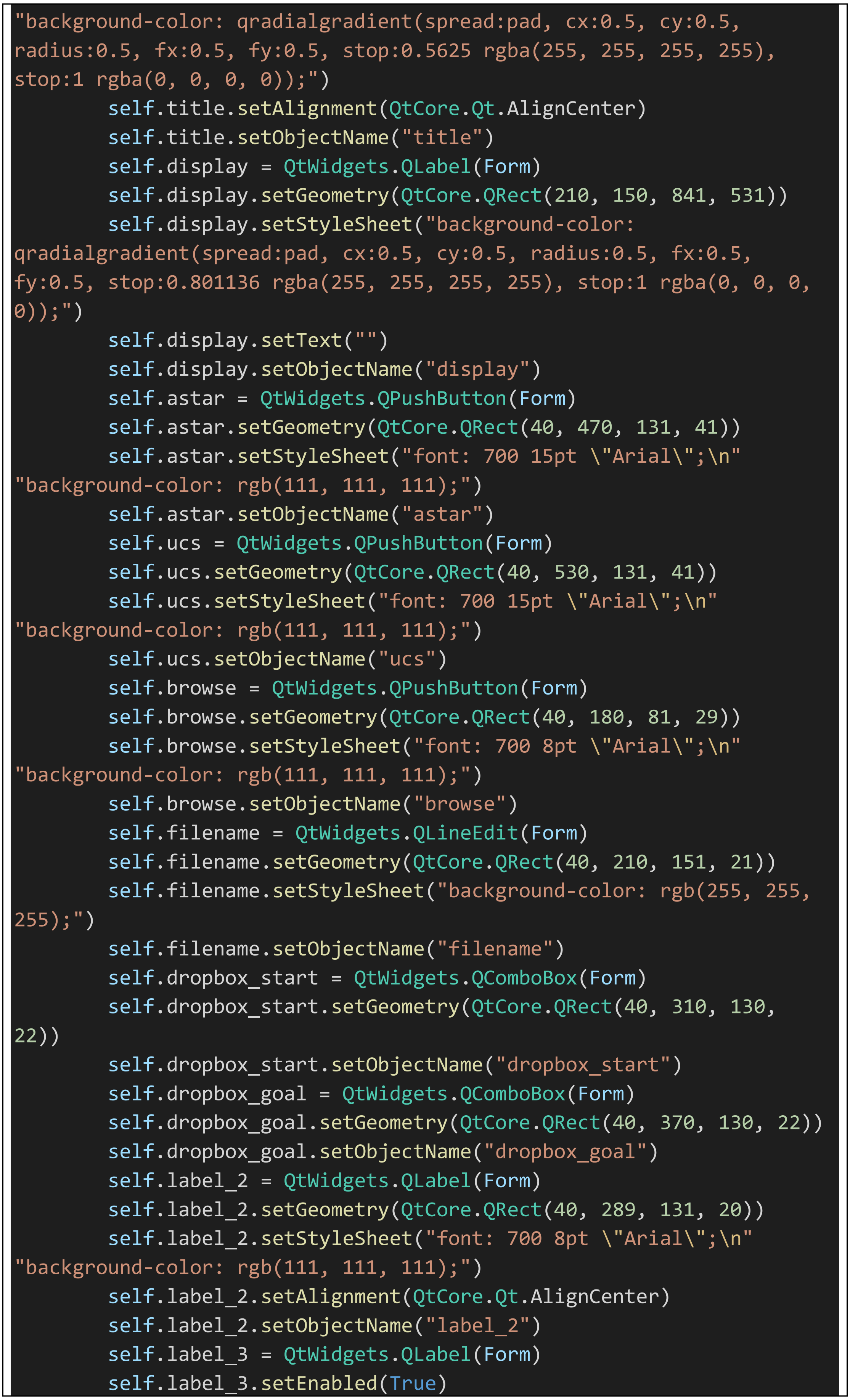
Berikut adalah langkah-langkah menjalankan program yang telah dibuat:

1. Program dapat dijalankan dengan melakukan ***cd src*** lalu ***python main.py***
2. Program akan meminta user untuk ***Browse*** untuk mendapatkan file yang ingin diuji.
3. User akan diminta untuk mengisi start dan goal node.
4. Jika tombol A* ditekan maka program akan menjalankan algoritma A* berdasarkan input yang telah didapatkan.
5. Jika tombol UCS ditekan maka program akan menjalankan algoritma UCS berdasarkan input yang telah didapatkan.
6. Setelah menekan salah satu tombol A* atau UCS, program akan menampilkan peta google map dengan path terpendek menurut algoritma yang dipilih.

BAB IV
KODE PROGRAM

1. File ui_GUI.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'GUI.ui'
#
# Created by: PyQt5 UI code generator 5.15.9
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(1080, 720)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(Form.sizePolicy().hasHeightForWidth())
        Form.setSizePolicy(sizePolicy)
        Form.setMinimumSize(QtCore.QSize(1080, 720))
        Form.setMaximumSize(QtCore.QSize(1080, 720))
        self.label = QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(0, 0, 1080, 720))
        self.label.setStyleSheet("border-image: url(:/newPrefix/images/background.jpg);")
        self.label.setText("")
        self.label.setObjectName("label")
        self.title = QtWidgets.QLabel(Form)
        self.title.setGeometry(QtCore.QRect(30, 30, 1021, 81))
        self.title.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.title.setStyleSheet("color: rgb(0, 0, 0);\n"
"font: 700 25pt \"Arial\";\n"
```



```
"background-color: qradialgradient(spread:pad, cx:0.5, cy:0.5,
radius:0.5, fx:0.5, fy:0.5, stop:0.5625 rgba(255, 255, 255, 255),
stop:1 rgba(0, 0, 0, 0));")
        self.title.setAlignment(QtCore.Qt.AlignCenter)
        self.title.setObjectName("title")
        self.display = QtWidgets.QLabel(Form)
        self.display.setGeometry(QtCore.QRect(210, 150, 841, 531))
        self.display.setStyleSheet("background-color:
qradialgradient(spread:pad, cx:0.5, cy:0.5, radius:0.5, fx:0.5,
fy:0.5, stop:0.801136 rgba(255, 255, 255, 255), stop:1 rgba(0, 0, 0,
0));")
        self.display.setText("")
        self.display.setObjectName("display")
        self.astar = QtWidgets.QPushButton(Form)
        self.astar.setGeometry(QtCore.QRect(40, 470, 131, 41))
        self.astar.setStyleSheet("font: 700 15pt \"Arial\";\n"
"background-color: rgb(111, 111, 111);")
        self.astar.setObjectName("astar")
        self.ucs = QtWidgets.QPushButton(Form)
        self.ucs.setGeometry(QtCore.QRect(40, 530, 131, 41))
        self.ucs.setStyleSheet("font: 700 15pt \"Arial\";\n"
"background-color: rgb(111, 111, 111);")
        self.ucs.setObjectName("ucs")
        self.browse = QtWidgets.QPushButton(Form)
        self.browse.setGeometry(QtCore.QRect(40, 180, 81, 29))
        self.browse.setStyleSheet("font: 700 8pt \"Arial\";\n"
"background-color: rgb(111, 111, 111);")
        self.browse.setObjectName("browse")
        self.filename = QtWidgets.QLineEdit(Form)
        self.filename.setGeometry(QtCore.QRect(40, 210, 151, 21))
        self.filename.setStyleSheet("background-color: rgb(255, 255,
255);")
        self.filename.setObjectName("filename")
        self.dropbox_start = QtWidgets.QComboBox(Form)
        self.dropbox_start.setGeometry(QtCore.QRect(40, 310, 130,
22))
        self.dropbox_start.setObjectName("dropbox_start")
        self.dropbox_goal = QtWidgets.QComboBox(Form)
        self.dropbox_goal.setGeometry(QtCore.QRect(40, 370, 130, 22))
        self.dropbox_goal.setObjectName("dropbox_goal")
        self.label_2 = QtWidgets.QLabel(Form)
        self.label_2.setGeometry(QtCore.QRect(40, 289, 131, 20))
        self.label_2.setStyleSheet("font: 700 8pt \"Arial\";\n"
"background-color: rgb(111, 111, 111);")
        self.label_2.setAlignment(QtCore.Qt.AlignCenter)
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(Form)
        self.label_3.setEnabled(True)
        self.label_3.setGeometry(QtCore.QRect(40, 349, 131, 20))
        self.label_3.setStyleSheet("font: 700 8pt \"Arial\";\n"
"background-color: rgb(111, 111, 111);")
        self.label_3.setAlignment(QtCore.Qt.AlignCenter)
        self.label_3.setObjectName("label_3")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.title.setText(_translate("Form", "Menentukan Lintasan
Terpendek"))
        self.astar.setText(_translate("Form", "A*"))
        self.ucs.setText(_translate("Form", "UCS"))
        self.browse.setText(_translate("Form", "Browse"))
        self.label_2.setText(_translate("Form", "Start"))
        self.label_3.setText(_translate("Form", "Goal"))
import resources_rc
```

2. File map_visual.py

```
# file: map_visual.py

import folium
import graph as g

def color(name, solution):
    # kalau starting point
    if name == solution[0][0]:
        color = 'green'
    else:
        # kalau di path
        if name in solution[0]:
            color = 'red'
        else:
            color = 'blue'
    return color

def visual_map(list_path, path_solution, isUCS):

    map =
folium.Map(location=[g.avg_lat(g.list_lat),g.avg_lon(g.list_lon)],zoo
m_start=15)

    # make markers
    for point in range(0, len(g.list_of_coordinates)):
        folium.Marker(g.list_of_coordinates[point],
popup=g.list_of_names[point],
icon=folium.Icon(color=color(g.list_of_names[point],
path_solution))).add_to(map)

    # make path
    fg = folium.FeatureGroup("Path")
    line = folium.vector_layers.PolyLine(list_path, color='red',
weight=10).add_to(fg)
    fg.add_to(map)

    # add lintasan terpendek
    solution = g.string_route(path_solution)
    solution_html = '''
                <h3 align="center" style="font-size:20px"><b>{}</b>
                </h3>
                '''.format(solution)
    map.get_root().html.add_child(folium.Element(solution_html))

    if isUCS:
        panjang_lintasan = "Panjang lintasan UCS : "
    else:
        panjang_lintasan = "Panjang lintasan A* : "
    panjang_lintasan += str(path_solution[1]) + " meter."

    panjang_lintasan_html = '''
                    <h3 align="center" style="font-
size:20px"><b>{}</b>
                    </h3>
                    '''.format(panjang_lintasan)
    map.get_root().html.add_child(folium.Element(panjang_lintasan_htm
l))

    map.add_child(folium.LayerControl())
    map.save(outfile='map.html')
```

3. File show_map.py

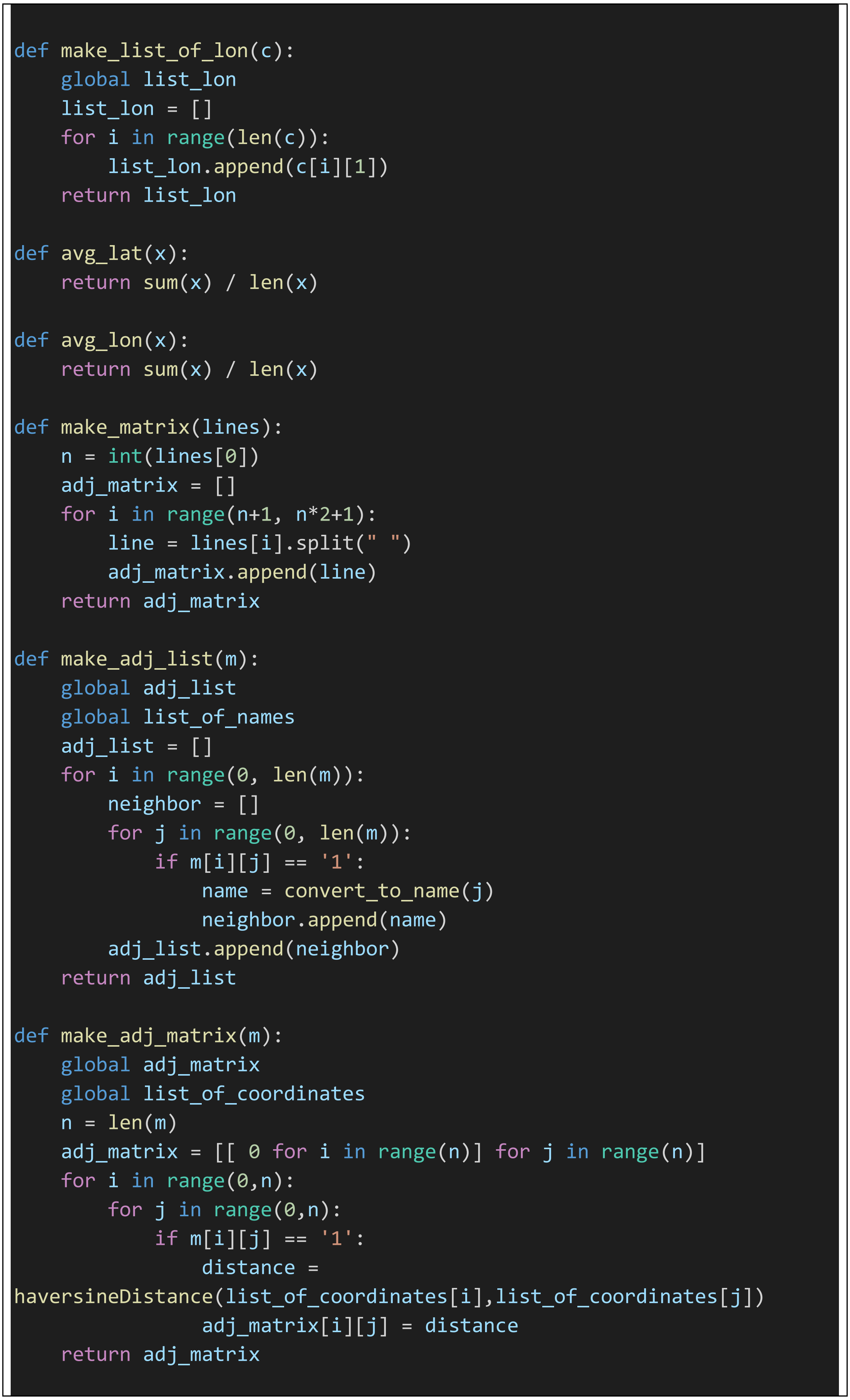
```
# file: show_map.py

from PyQt5.QtCore import QUrl
from PyQt5 import QtCore
from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout
from PyQt5.QtWebEngineWidgets import QWebEngineView
import os

class MainWindow(QMainWindow):
    # Define the file path as a class variable
    html_file_path = os.getcwd() + "/map.html"

    def __init__(self):
        super().__init__()

        # Create a QWidget instance to hold the QWebEngineView widget
        widget = QWidget(self)
        self.setCentralWidget(widget)

        self.setMinimumSize(QtCore.QSize(1080, 720))
        self.setMaximumSize(QtCore.QSize(1080, 720))

        # Create a QVBoxLayout instance to hold the QWebEngineView
widget
        layout = QVBoxLayout(widget)

        # Create a QWebEngineView widget and add it to the layout
        self.webview = QWebEngineView()
        self.webview.setGeometry(QtCore.QRect(210, 150, 841, 531))
        self.webview.setMinimumSize(QtCore.QSize(841, 531))
        self.webview.setMaximumSize(QtCore.QSize(841, 531))
        layout.addWidget(self.webview)

        # Load the HTML file into the QWebEngineView widget
        self.url = QUrl.fromLocalFile(MainWindow.html_file_path)
        print("Was here")
```

4. File GUI_visual.py

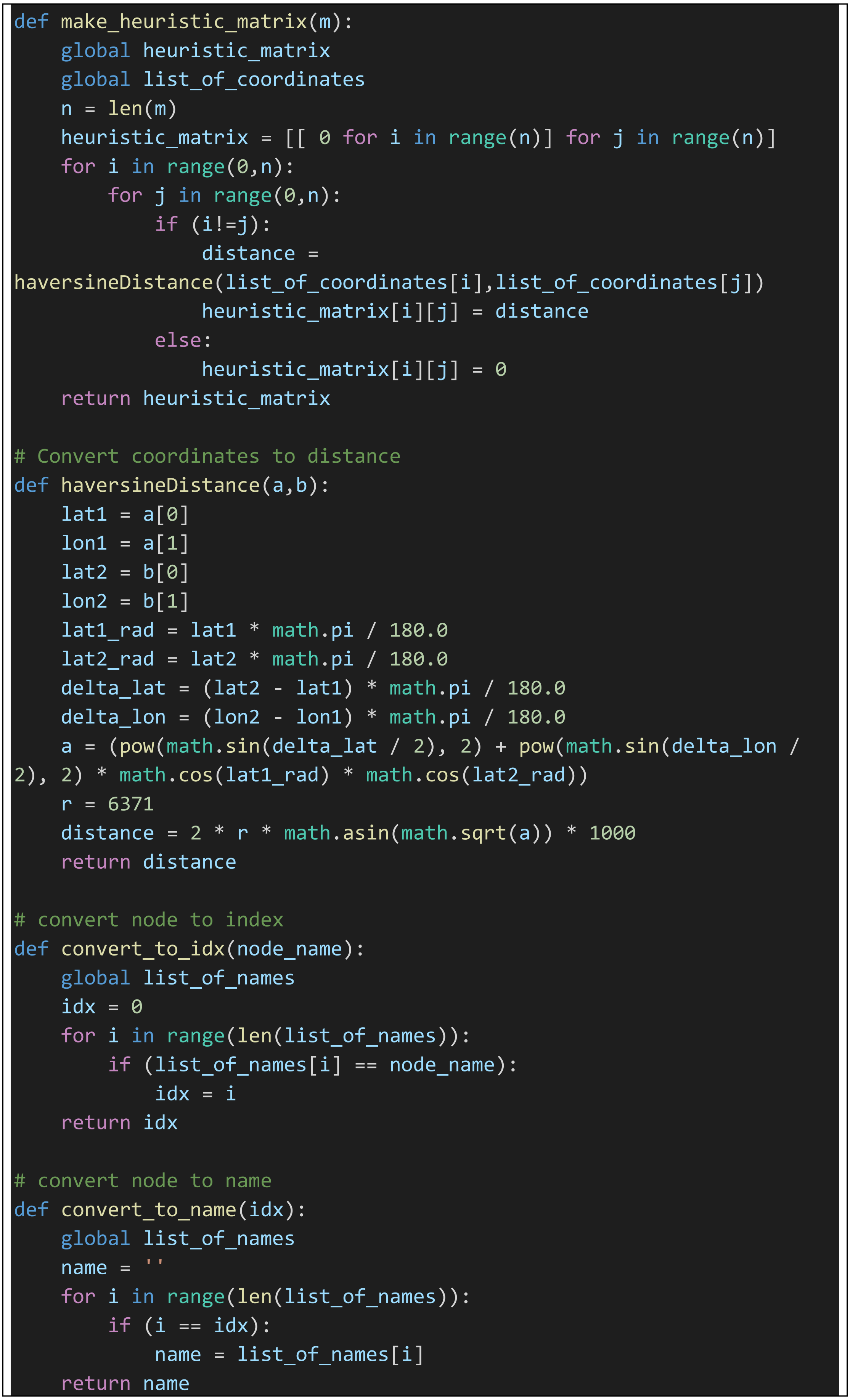
```
# File GUI_visual.py

from PyQt5.QtWidgets import QWidget , QFileDialog
from ui_GUI import Ui_Form
from show_map import MainWindow as map
from PyQt5 import QtCore
import graph as g
import map_visual as mv

# Pemanggilan GUI
class ShortestPathFinder(map, QWidget, Ui_Form):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.browse.clicked.connect(self.select_file_button_clicked)
        self.astar.clicked.connect(self.astar_clicked)
        self.ucs.clicked.connect(self.ucs_clicked)
        self.isMapSelected = False
    def select_file_button_clicked(self):
        file_name,_ = QFileDialog.getOpenFileName(self, "Open File",
                                 "../test",
                                 "Text(*.txt);;Images (*.png *.xpm
*.jpg);;All files(*.*)")
        if((file_name == "")):
            return

        g.initialize(file_name)

        self.filename.setText(file_name)
        self.dropbox_start.clear()
        self.dropbox_goal.clear()
        _translate = QtCore.QCoreApplication.translate

        for i in range (len(g.list_of_names)):
            self.dropbox_start.addItem("")
            self.dropbox_start.setItemText(i, _translate("Form",
g.list_of_names[i]))
            self.dropbox_goal.addItem("")
            self.dropbox_goal.setItemText(i, _translate("Form",
g.list_of_names[i]))

        self.isMapSelected = True

    def astar_clicked(self):
        if (not self.isMapSelected):
            return

        self.startNode = self.dropbox_start.currentText()
        self.goalNode = self.dropbox_goal.currentText()
        if self.dropbox_start.currentIndex() >
self.dropbox_goal.currentIndex():
            self.startNode, self.goalNode = self.goalNode,
self.startNode

        self.label.lower()
        self.display.lower()
        self.webview.setGeometry(QtCore.QRect(210, 150, 841, 531))

        self.path_solution = g.astar(self.startNode, self.goalNode)
        self.list_path = g.path_coords(self.path_solution)

        mv.visual_map(self.list_path, self.path_solution, False)

        self.webview.load(self.url)
    def ucs_clicked(self):
        if (not self.isMapSelected):
            return

        self.startNode = self.dropbox_start.currentText()
        self.goalNode = self.dropbox_goal.currentText()
        if self.dropbox_start.currentIndex() > 
self.dropbox_goal.currentIndex():
            self.startNode, self.goalNode = self.goalNode, 
self.startNode

        self.label.lower()
        self.display.lower()
        self.webview.setGeometry(QtCore.QRect(210, 150, 841, 531))

        self.path_solution = g.ucs(self.startNode, self.goalNode)
        self.list_path = g.path_coords(self.path_solution)

        mv.visual_map(self.list_path, self.path_solution, True)

        self.webview.load(self.url)
```

5. File graph.py

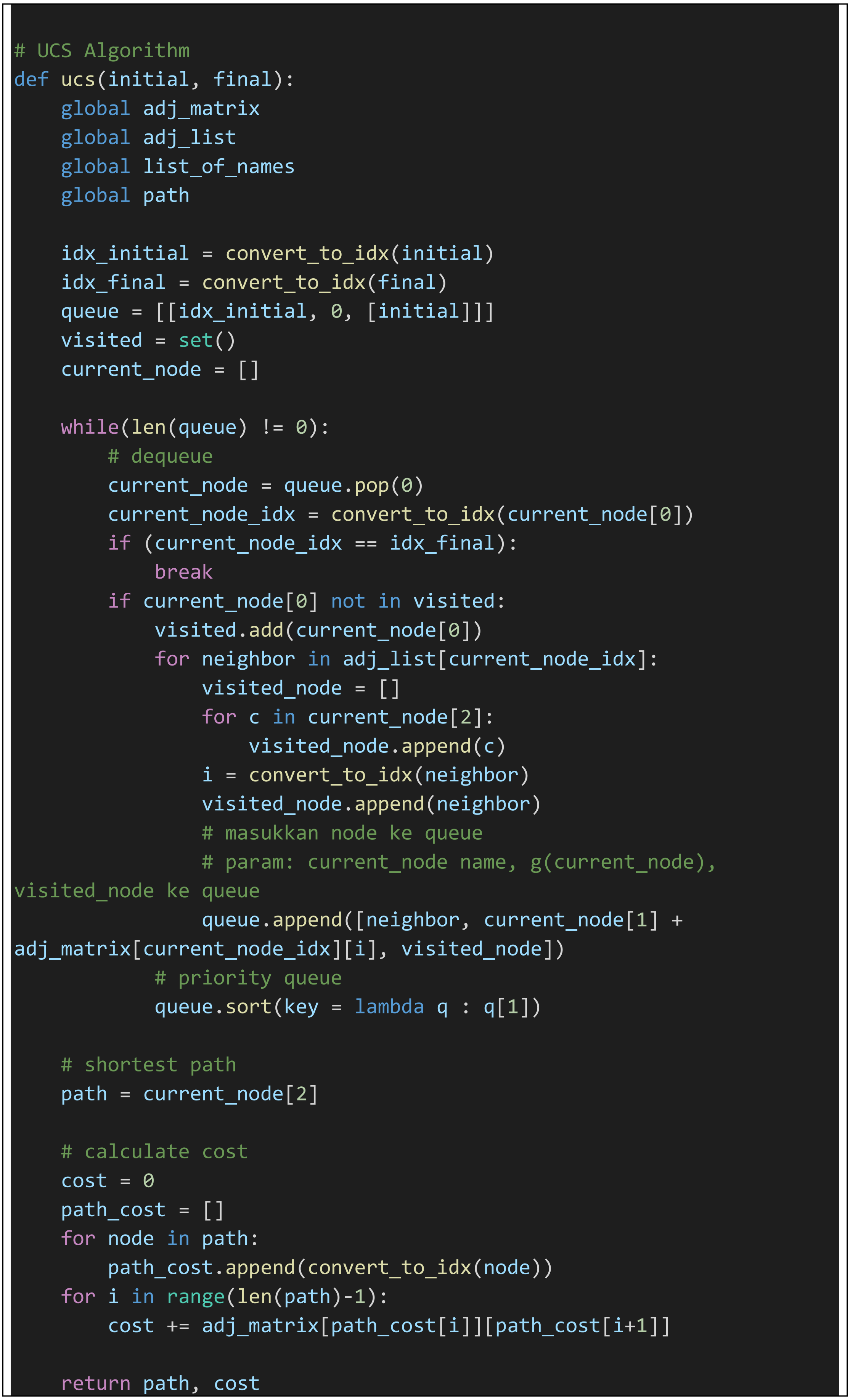
```
# File: graph.py

import math

def make_coordinates(line):
    global list_of_coordinates
    global list_of_names
    n = int(line[0])
    list_of_coordinates = []
    list_of_names = []
    for i in range(1, n+1):
        coordinates = line[i].split(" ")
        coords_list = [float(coordinates[1]), float(coordinates[2])]
        list_of_coordinates.append(coords_list)
        list_of_names.append(coordinates[0])
    # print(list_of_names)
    return list_of_coordinates, list_of_names

def make_list_of_lat(c):
    global list_lat
    list_lat = []
    for i in range(len(c)):
        list_lat.append(c[i][0])
    return list_lat
```



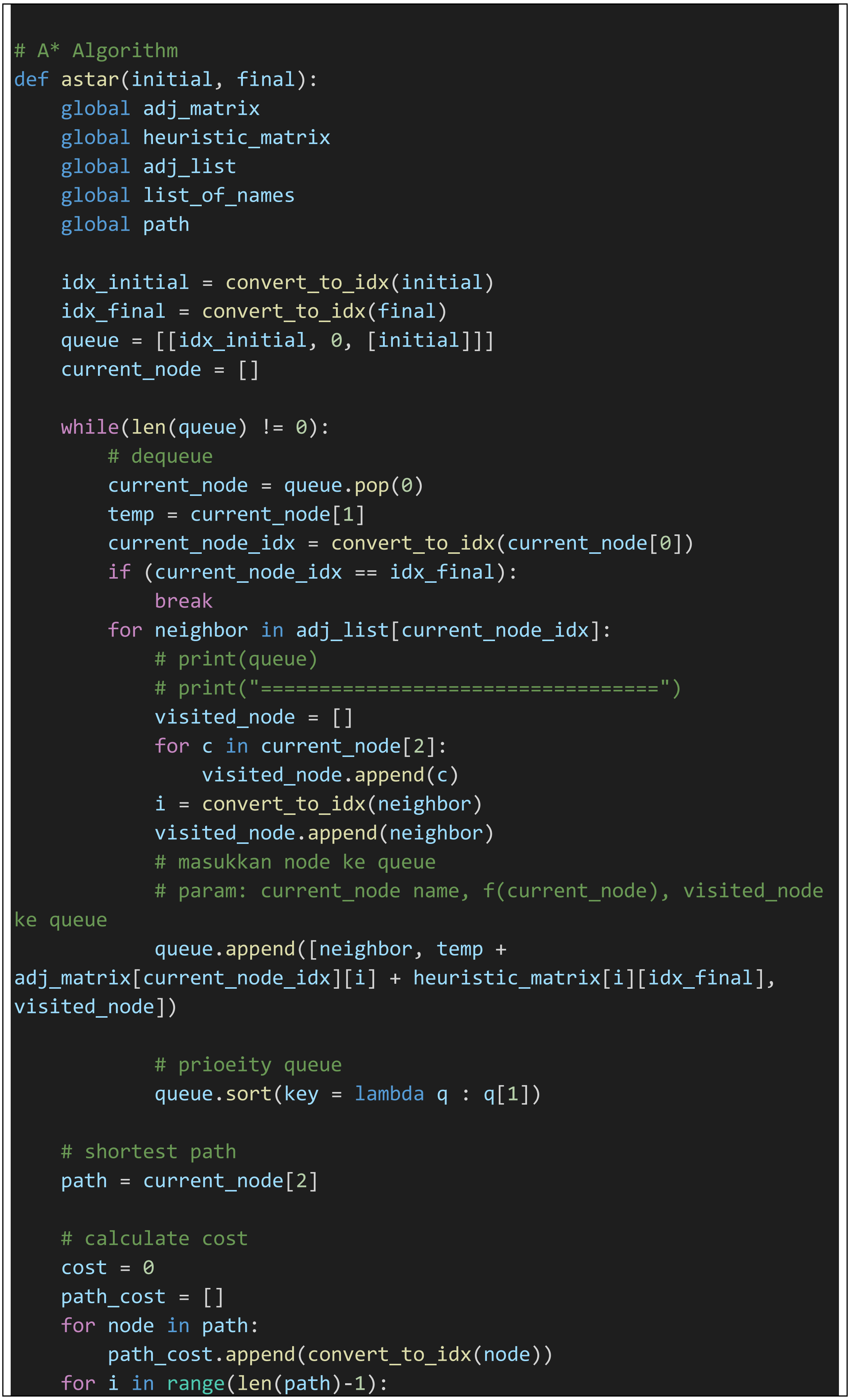
```
def make_list_of_lon(c):
    global list_lon
    list_lon = []
    for i in range(len(c)):
        list_lon.append(c[i][1])
    return list_lon

def avg_lat(x):
    return sum(x) / len(x)

def avg_lon(x):
    return sum(x) / len(x)

def make_matrix(lines):
    n = int(lines[0])
    adj_matrix = []
    for i in range(n+1, n*2+1):
        line = lines[i].split(" ")
        adj_matrix.append(line)
    return adj_matrix

def make_adj_list(m):
    global adj_list
    global list_of_names
    adj_list = []
    for i in range(0, len(m)):
        neighbor = []
        for j in range(0, len(m)):
            if m[i][j] == '1':
                name = convert_to_name(j)
                neighbor.append(name)
        adj_list.append(neighbor)
    return adj_list

def make_adj_matrix(m):
    global adj_matrix
    global list_of_coordinates
    n = len(m)
    adj_matrix = [[ 0 for i in range(n)] for j in range(n)]
    for i in range(0,n):
        for j in range(0,n):
            if m[i][j] == '1':
                distance =
haversineDistance(list_of_coordinates[i],list_of_coordinates[j])
                adj_matrix[i][j] = distance
    return adj_matrix
```



```
def make_heuristic_matrix(m):
    global heuristic_matrix
    global list_of_coordinates
    n = len(m)
    heuristic_matrix = [[ 0 for i in range(n)] for j in range(n)]
    for i in range(0,n):
        for j in range(0,n):
            if (i!=j):
                distance =
haversineDistance(list_of_coordinates[i],list_of_coordinates[j])
                heuristic_matrix[i][j] = distance
            else:
                heuristic_matrix[i][j] = 0
    return heuristic_matrix

# Convert coordinates to distance
def haversineDistance(a,b):
    lat1 = a[0]
    lon1 = a[1]
    lat2 = b[0]
    lon2 = b[1]
    lat1_rad = lat1 * math.pi / 180.0
    lat2_rad = lat2 * math.pi / 180.0
    delta_lat = (lat2 - lat1) * math.pi / 180.0
    delta_lon = (lon2 - lon1) * math.pi / 180.0
    a = (pow(math.sin(delta_lat / 2), 2) + pow(math.sin(delta_lon /
2), 2) * math.cos(lat1_rad) * math.cos(lat2_rad))
    r = 6371
    distance = 2 * r * math.asin(math.sqrt(a)) * 1000
    return distance

# convert node to index
def convert_to_idx(node_name):
    global list_of_names
    idx = 0
    for i in range(len(list_of_names)):
        if (list_of_names[i] == node_name):
            idx = i
    return idx

# convert node to name
def convert_to_name(idx):
    global list_of_names
    name = ''
    for i in range(len(list_of_names)):
        if (i == idx):
            name = list_of_names[i]
    return name
```



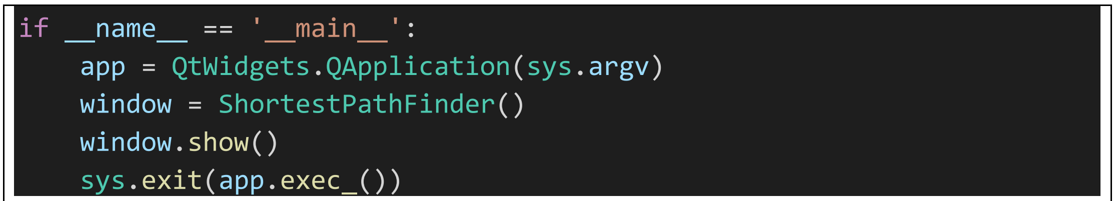
```
# UCS Algorithm
def ucs(initial, final):
    global adj_matrix
    global adj_list
    global list_of_names
    global path

    idx_initial = convert_to_idx(initial)
    idx_final = convert_to_idx(final)
    queue = [[idx_initial, 0, [initial]]]
    visited = set()
    current_node = []

    while(len(queue) != 0):
        # dequeue
        current_node = queue.pop(0)
        current_node_idx = convert_to_idx(current_node[0])
        if (current_node_idx == idx_final):
            break
        if current_node[0] not in visited:
            visited.add(current_node[0])
            for neighbor in adj_list[current_node_idx]:
                visited_node = []
                for c in current_node[2]:
                    visited_node.append(c)
                i = convert_to_idx(neighbor)
                visited_node.append(neighbor)
                # masukkan node ke queue
                # param: current_node name, g(current_node),
visited_node ke queue
                queue.append([neighbor, current_node[1] +
adj_matrix[current_node_idx][i], visited_node])
            # priority queue
            queue.sort(key = lambda q : q[1])

    # shortest path
    path = current_node[2]

    # calculate cost
    cost = 0
    path_cost = []
    for node in path:
        path_cost.append(convert_to_idx(node))
    for i in range(len(path)-1):
        cost += adj_matrix[path_cost[i]][path_cost[i+1]]

    return path, cost
```



```
# A* Algorithm
def astar(initial, final):
    global adj_matrix
    global heuristic_matrix
    global adj_list
    global list_of_names
    global path

    idx_initial = convert_to_idx(initial)
    idx_final = convert_to_idx(final)
    queue = [[idx_initial, 0, [initial]]]
    current_node = []

    while(len(queue) != 0):
        # dequeue
        current_node = queue.pop(0)
        temp = current_node[1]
        current_node_idx = convert_to_idx(current_node[0])
        if (current_node_idx == idx_final):
            break
        for neighbor in adj_list[current_node_idx]:
            # print(queue)
            # print("==================================")
            visited_node = []
            for c in current_node[2]:
                visited_node.append(c)
            i = convert_to_idx(neighbor)
            visited_node.append(neighbor)
            # masukkan node ke queue
            # param: current_node name, f(current_node), visited_node
ke queue
            queue.append([neighbor, temp +
adj_matrix[current_node_idx][i] + heuristic_matrix[i][idx_final],
visited_node])

            # prioeity queue
            queue.sort(key = lambda q : q[1])

    # shortest path
    path = current_node[2]

    # calculate cost
    cost = 0
    path_cost = []
    for node in path:
        path_cost.append(convert_to_idx(node))
    for i in range(len(path)-1):
        cost += adj_matrix[path_cost[i]][path_cost[i+1]]

    return path, cost

def path_coords(path):
    global list_of_names
    global list_of_coordinates
    global list_of_path_coords
    list_of_path_coords = []
    for node in path[0]:
        list_of_path_coords.append(list_of_coordinates[convert_to_idx
(node)])
    return list_of_path_coords

# get string route
def string_route(solution):
    solution_list = []
    for i in range(len(solution[0])):
        if (i == (len(solution[0])-1)):
            solution_list.append(solution[0][i])
        else:
            solution_list.append(solution[0][i]+" -> ")
    solution_list = ' '.join([str(elem) for elem in solution_list])
    return "Lintasan terpendek : " + (solution_list)

# initialize awal
def initialize(file_name):
    # data_folder = "../test/"
    file_to_open = file_name
    f = open(file_to_open, "r")
    lines = f.read().splitlines()
    coordinates = make_coordinates(lines)[0]
    list_lat = make_list_of_lat(coordinates)
    list_lon = make_list_of_lon(coordinates)
    node_names = make_coordinates(lines)[1]
    matrix = make_matrix(lines)
    adj_list = make_adj_list(matrix)
    adj_matrix = make_adj_matrix(matrix)
    heur_matrix = make_heuristic_matrix(matrix)
```

6. File main.py

```
# file: main.py

from GUI_visual import ShortestPathFinder
import PyQt5.QtWidgets as QtWidgets
import sys
```



```python
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = ShortestPathFinder()
    window.show()
    sys.exit(app.exec_())
```

BAB V

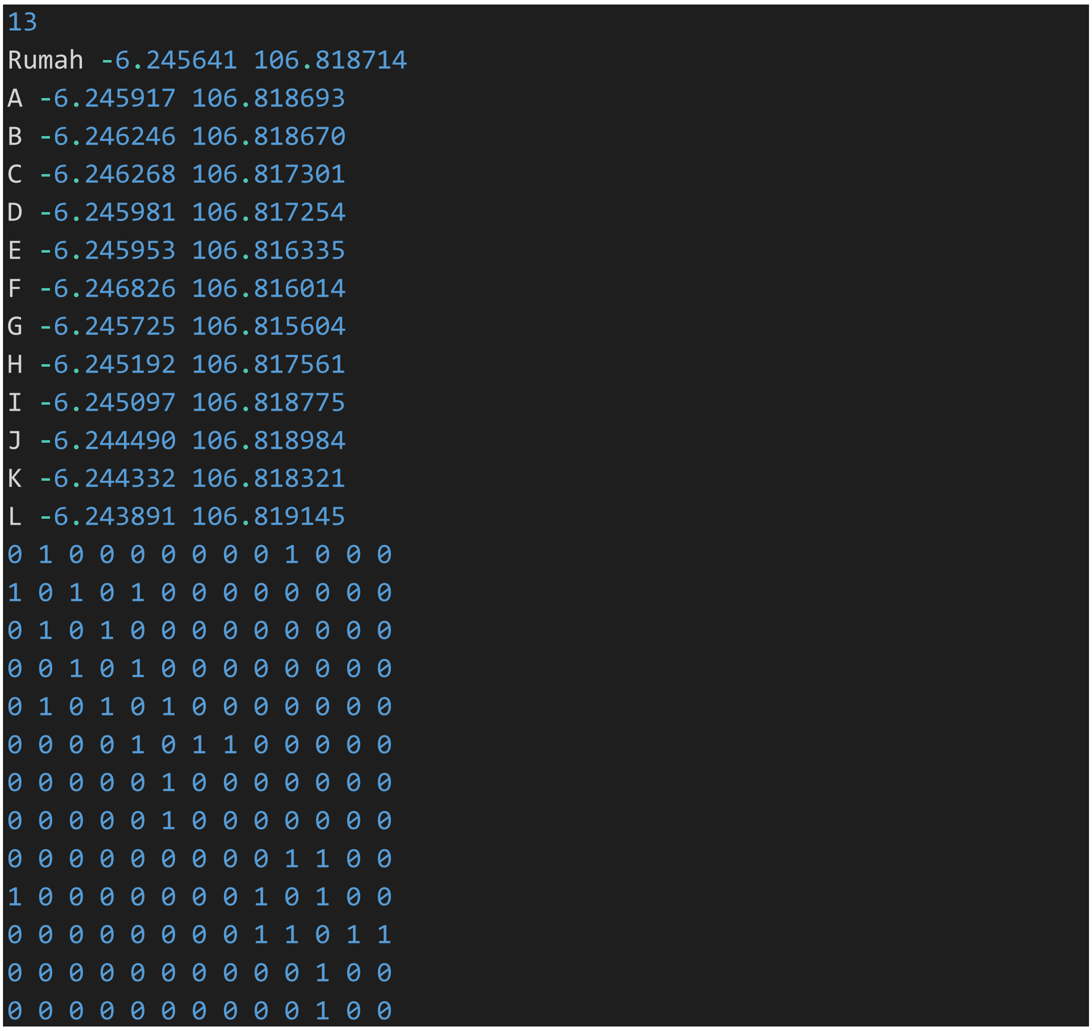
INPUT DAN OUTPUT

1. jakarta.txt

   Input:

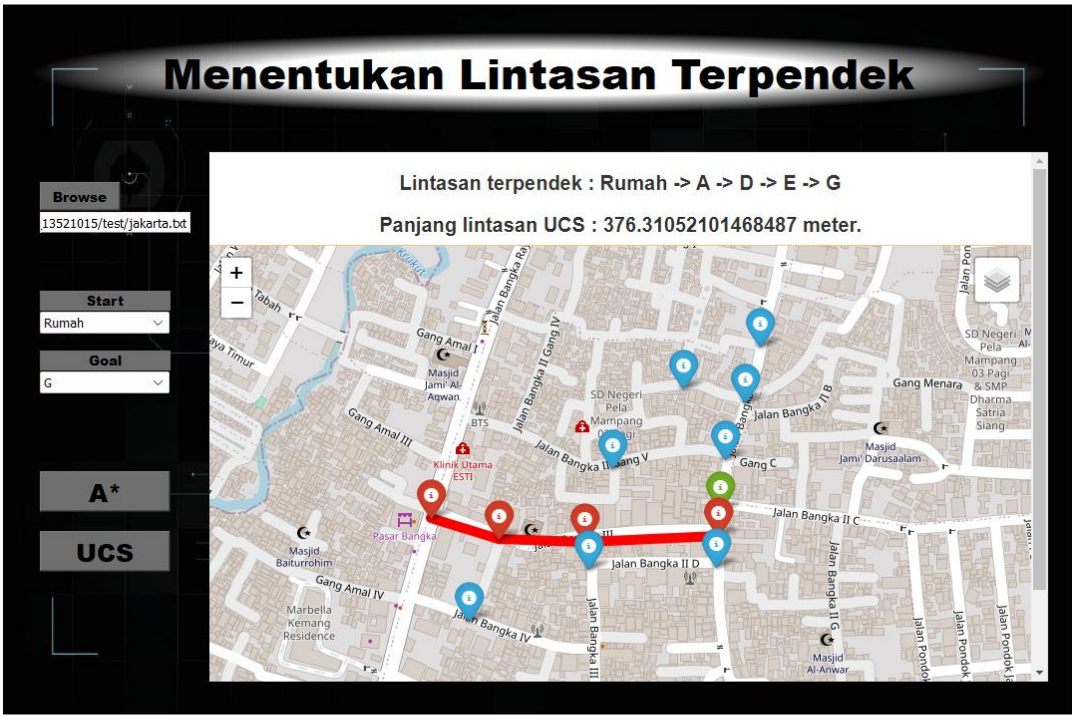
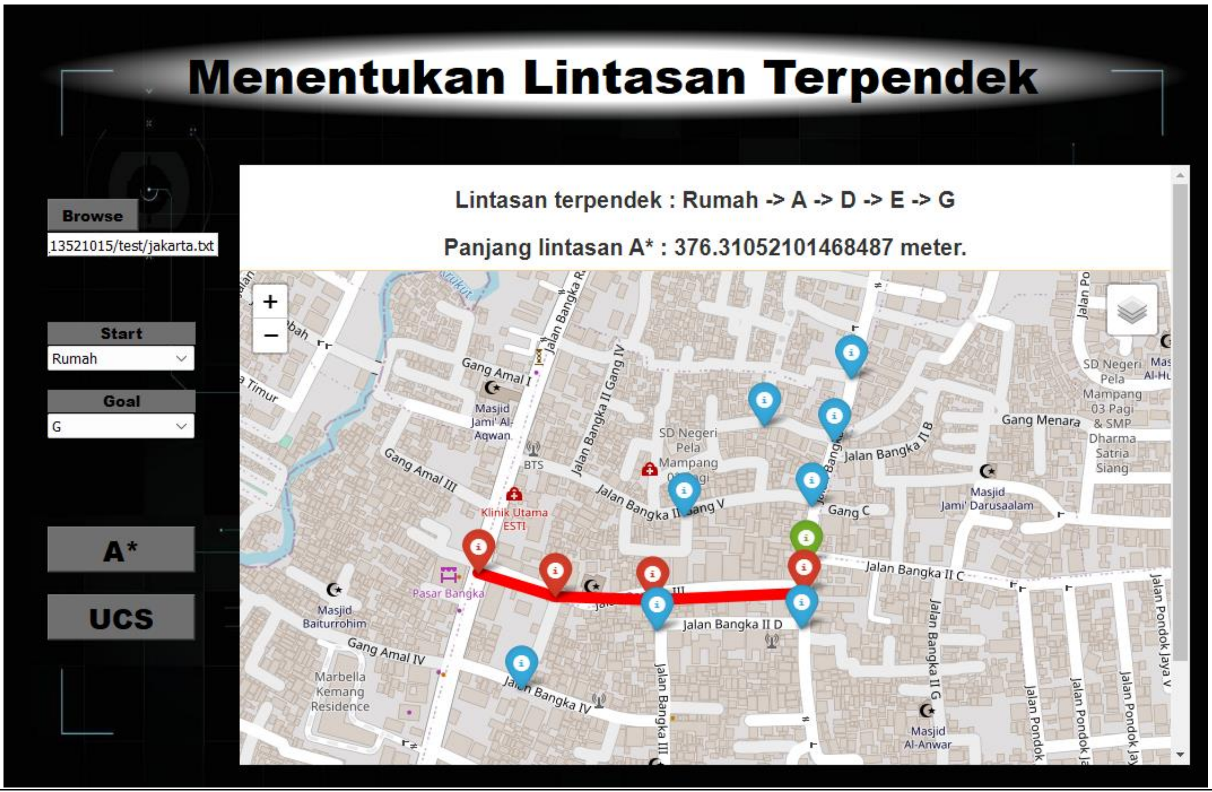
   Node start = Rumah

   Node goal = G



```
13
Rumah -6.245641 106.818714
A -6.245917 106.818693
B -6.246246 106.818670
C -6.246268 106.817301
D -6.245981 106.817254
E -6.245953 106.816335
F -6.246826 106.816014
G -6.245725 106.815604
H -6.245192 106.817561
I -6.245097 106.818775
J -6.244490 106.818984
K -6.244332 106.818321
L -6.243891 106.819145
0 1 0 0 0 0 0 0 0 1 0 0 0
1 0 1 0 1 0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 1 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 1 0 1 1 0 0 0 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 0 0 0 0 0 0 0 0 1 1 0 0
1 0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 1 1 0 1 1
0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0
```

Output:

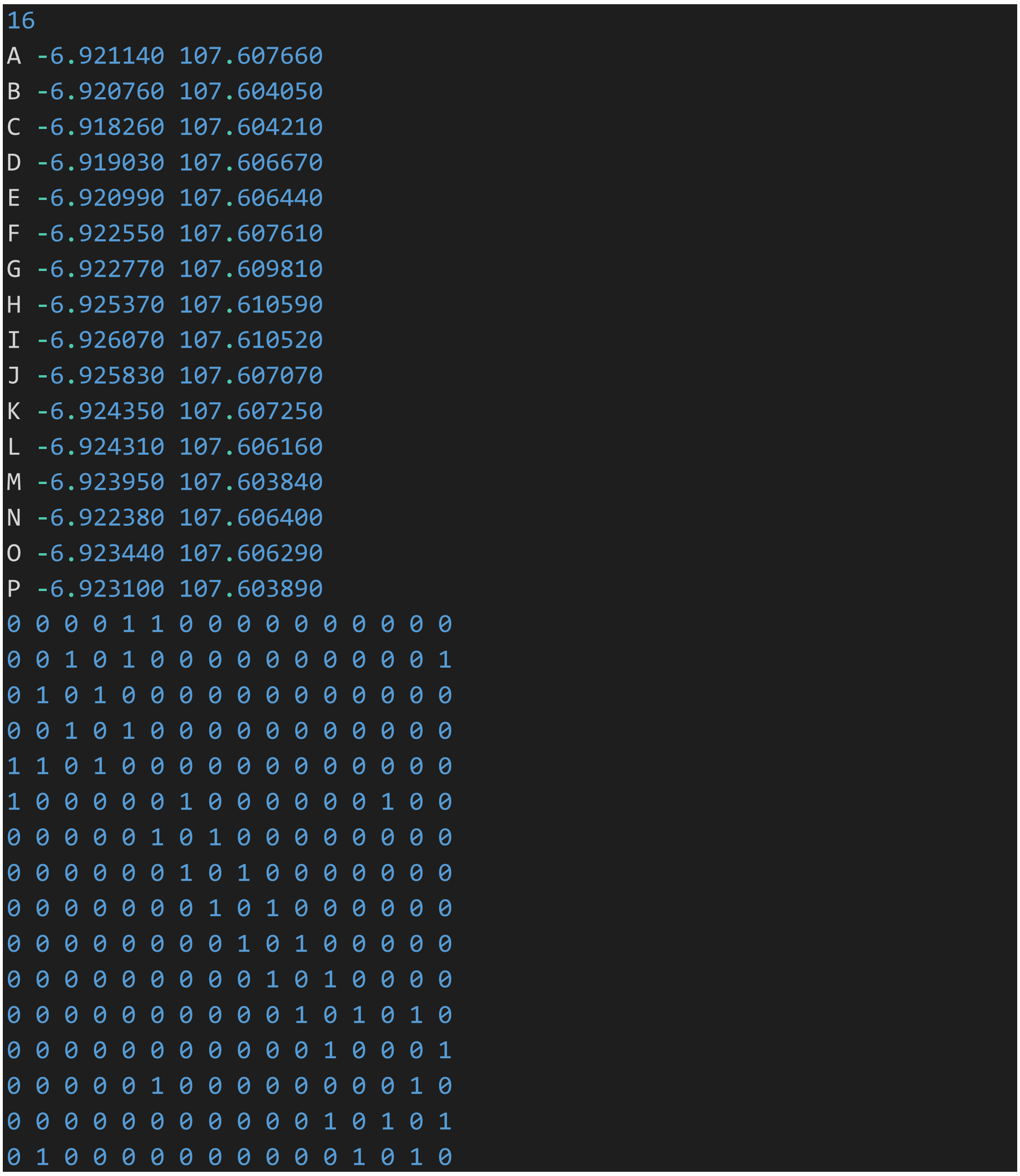
UCS
 Lintasan terpendek : Rumah -> A -> D -> E -> G Panjang lintasan UCS : 376.31052101468487 meter. Menentukan Lintasan Terpendek Browse 13521015/test/jakarta.txt Start Rumah Goal G A* UCS
A*
 Lintasan terpendek : Rumah -> A -> D -> E -> G Panjang lintasan A* : 376.31052101468487 meter. Menentukan Lintasan Terpendek Browse 13521015/test/jakarta.txt Start Rumah Goal G A* UCS

2. alun.txt

Input:

Node start = C

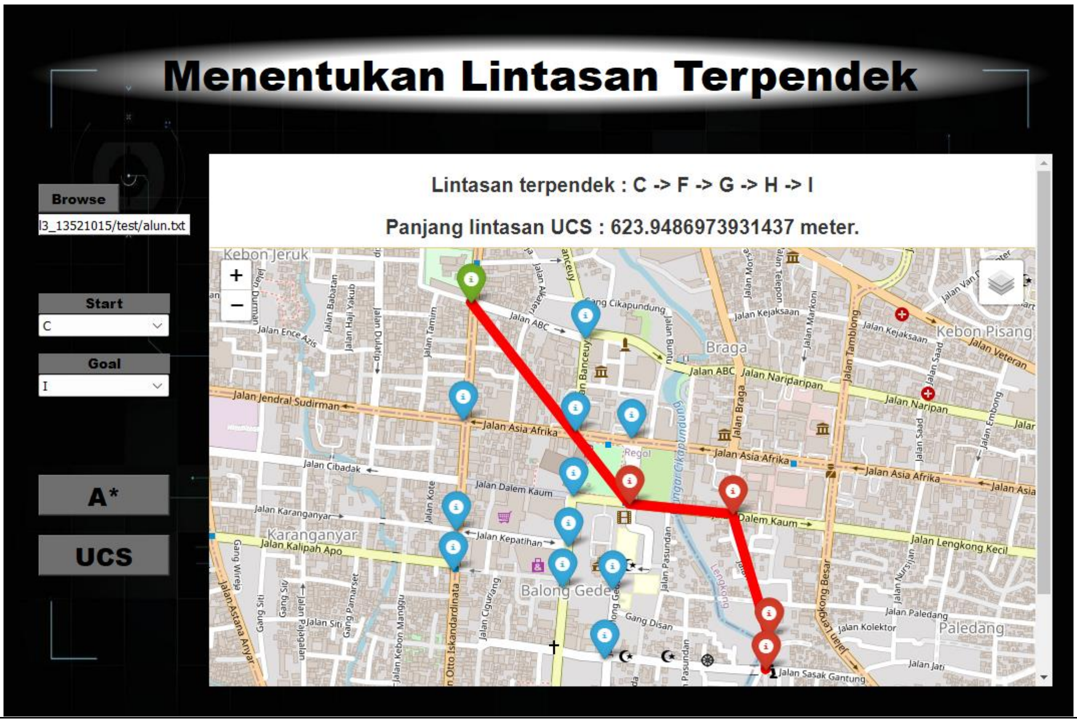
Node goal = I



```
16
A -6.921140 107.607660
B -6.920760 107.604050
C -6.918260 107.604210
D -6.919030 107.606670
E -6.920990 107.606440
F -6.922550 107.607610
G -6.922770 107.609810
H -6.925370 107.610590
I -6.926070 107.610520
J -6.925830 107.607070
K -6.924350 107.607250
L -6.924310 107.606160
M -6.923950 107.603840
N -6.922380 107.606400
O -6.923440 107.606290
P -6.923100 107.603890
0 0 0 0 1 1 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 1
0 1 0 1 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
1 1 0 1 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 1 0 0
0 0 0 0 0 1 0 1 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 1 0 1 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 1 0 1 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 1 0 0 0 1
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 0 1 0 1 0 1
0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

Output:

UCS



Menentukan Lintasan Terpendek

l3_13521015/test/alun.txt

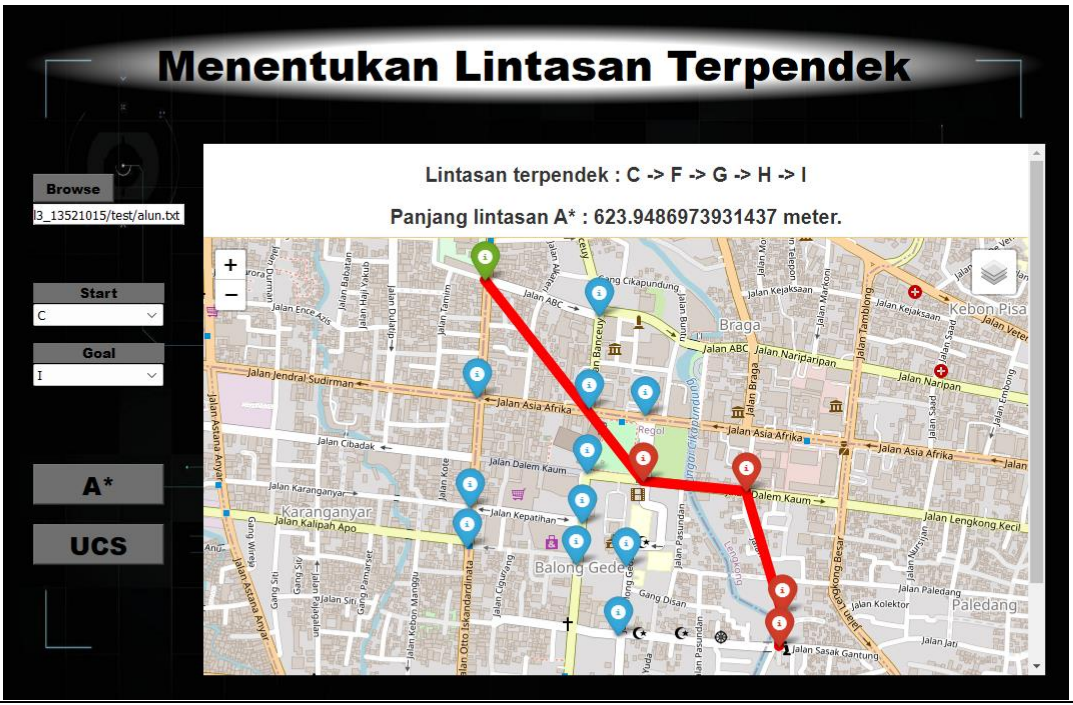
Start: C

Goal: I

Lintasan terpendek : C -> F -> G -> H -> I

Panjang lintasan UCS : 623.9486973931437 meter.

A*



Menentukan Lintasan Terpendek

l3_13521015/test/alun.txt

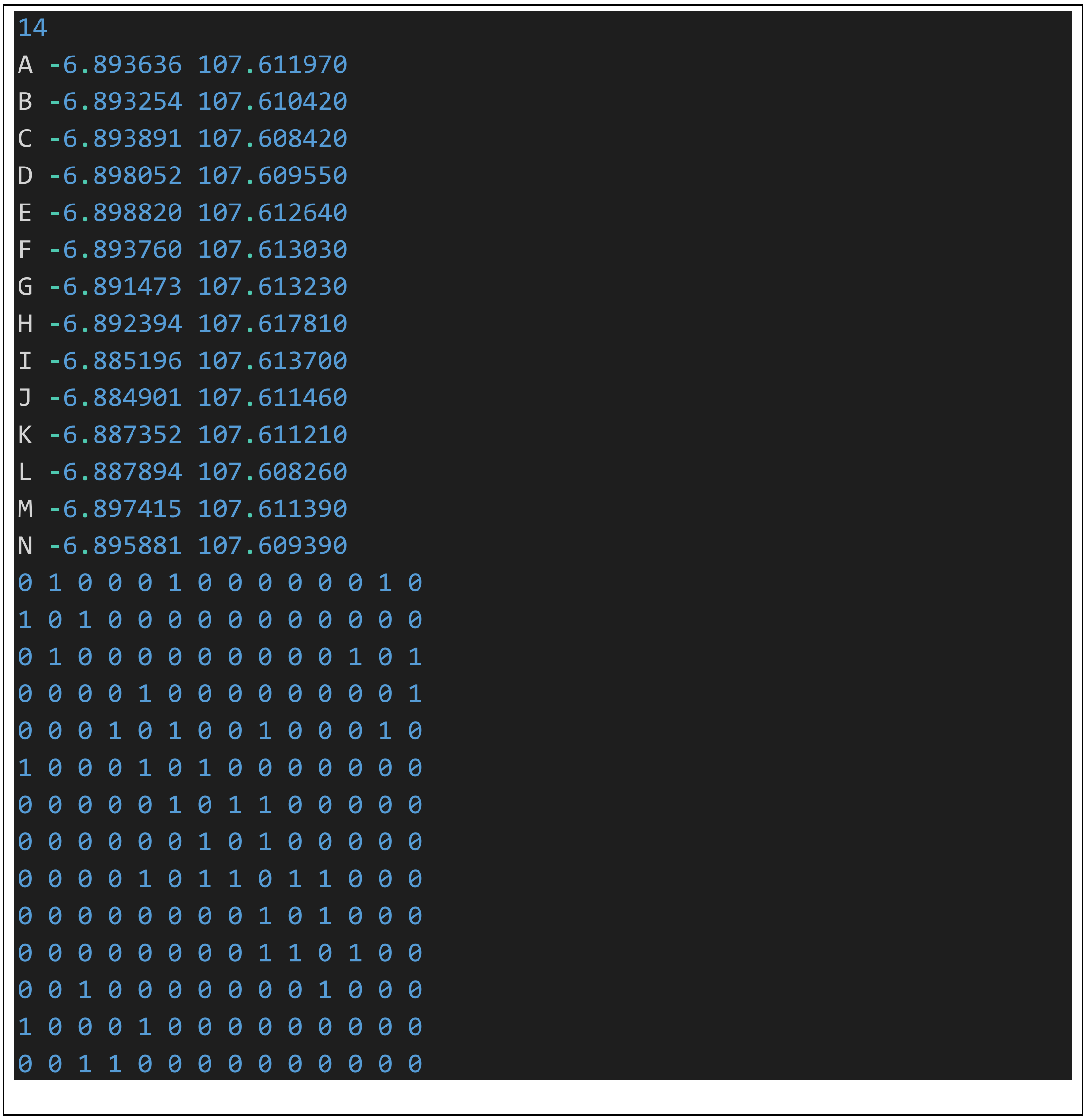
Start: C

Goal: I

Lintasan terpendek : C -> F -> G -> H -> I

Panjang lintasan A* : 623.9486973931437 meter.

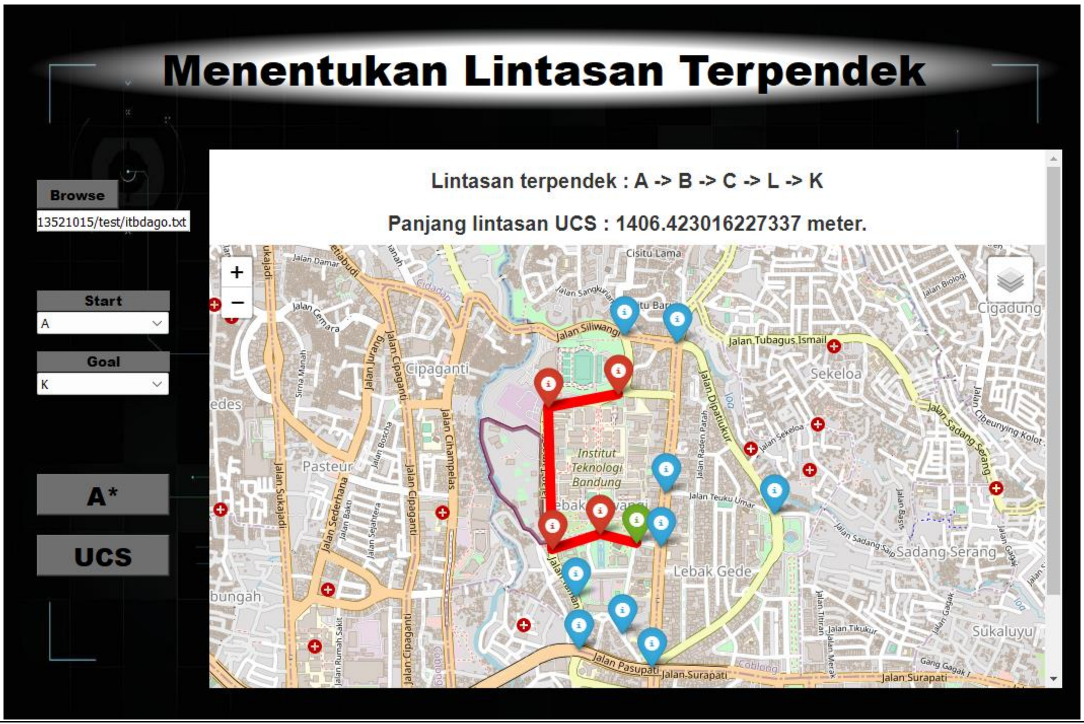
3. itbdago.txt
   Node start = A
   Node goal = K



```
14
A -6.893636 107.611970
B -6.893254 107.610420
C -6.893891 107.608420
D -6.898052 107.609550
E -6.898820 107.612640
F -6.893760 107.613030
G -6.891473 107.613230
H -6.892394 107.617810
I -6.885196 107.613700
J -6.884901 107.611460
K -6.887352 107.611210
L -6.887894 107.608260
M -6.897415 107.611390
N -6.895881 107.609390
0 1 0 0 0 1 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 1 0 1
0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 1 0 1 0 0 1 0 0 0 1 0
1 0 0 0 1 0 1 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 1 0 1 0 0 0 0 0
0 0 0 0 1 0 1 1 0 1 1 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 1 1 0 1 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0
```

Output:

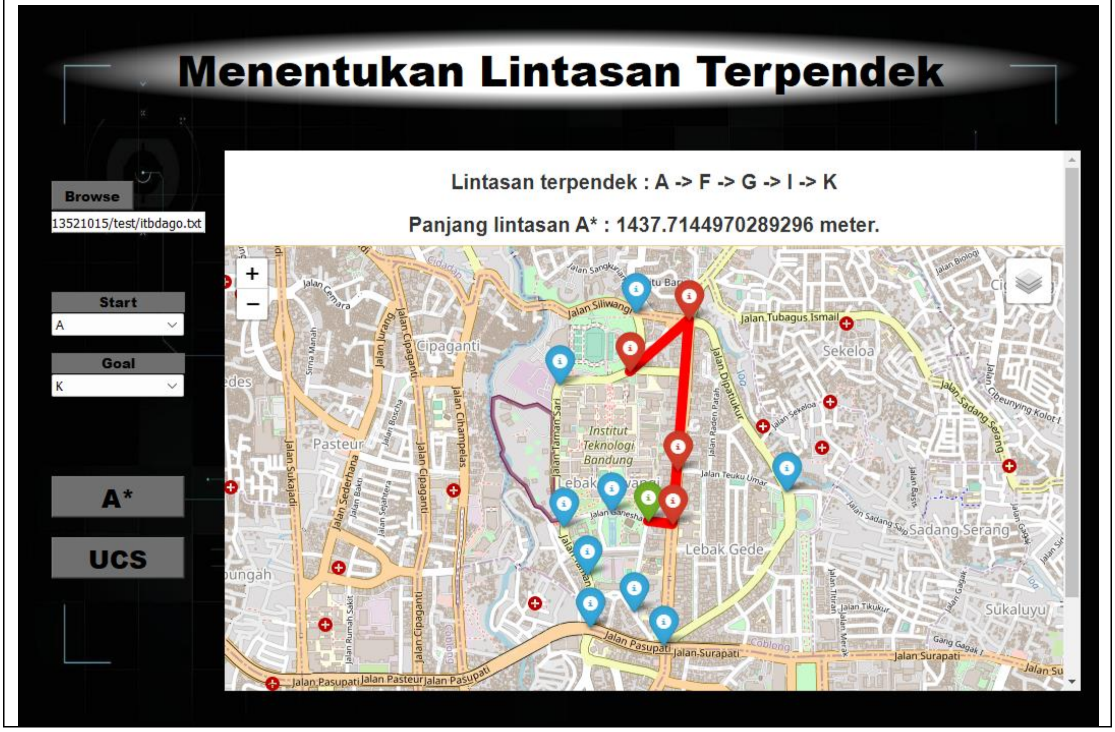
UCS



Lintasan terpendek : A -> B -> C -> L -> K

Panjang lintasan UCS : 1406.423016227337 meter.

A*



Lintasan terpendek : A -> F -> G -> I -> K

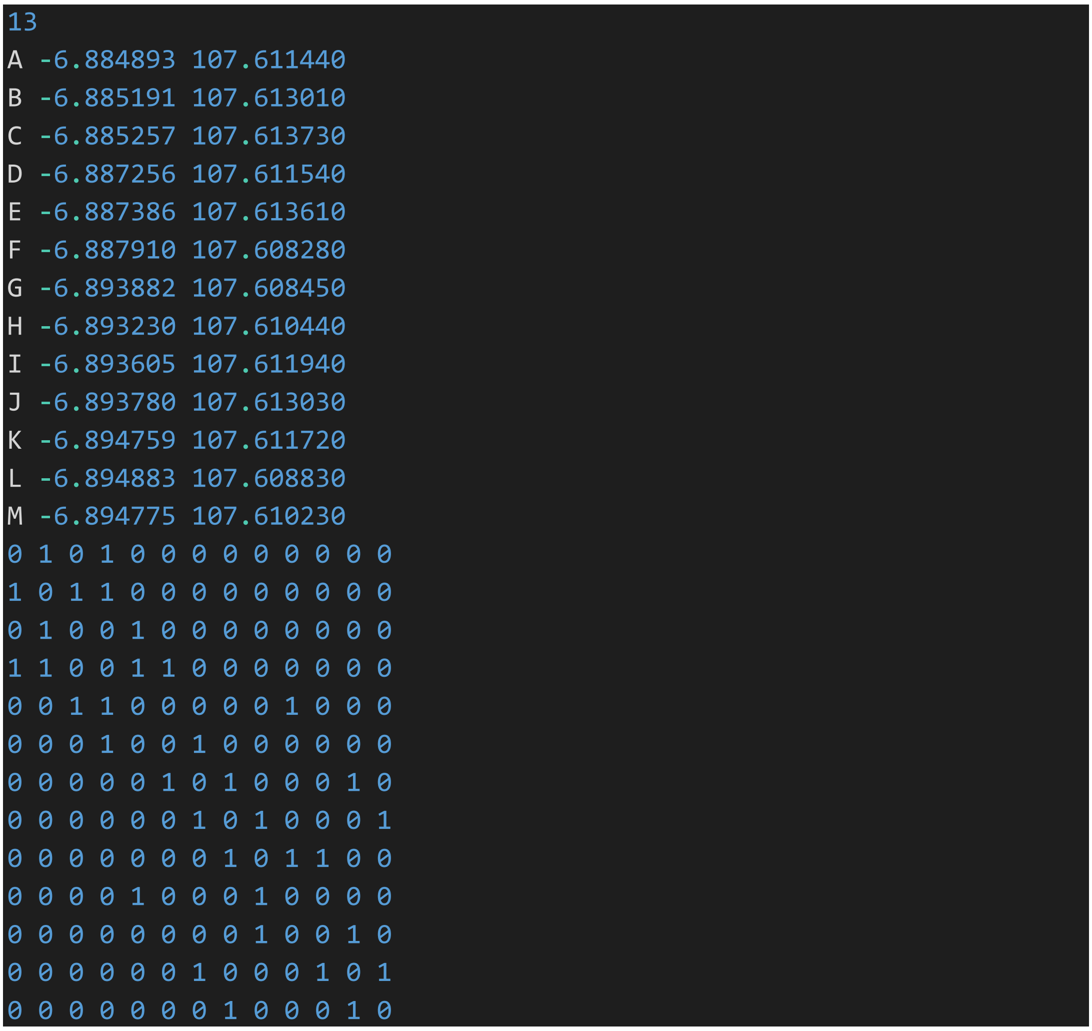
Panjang lintasan A* : 1437.7144970289296 meter.

4. itbgane.txt

   Input:

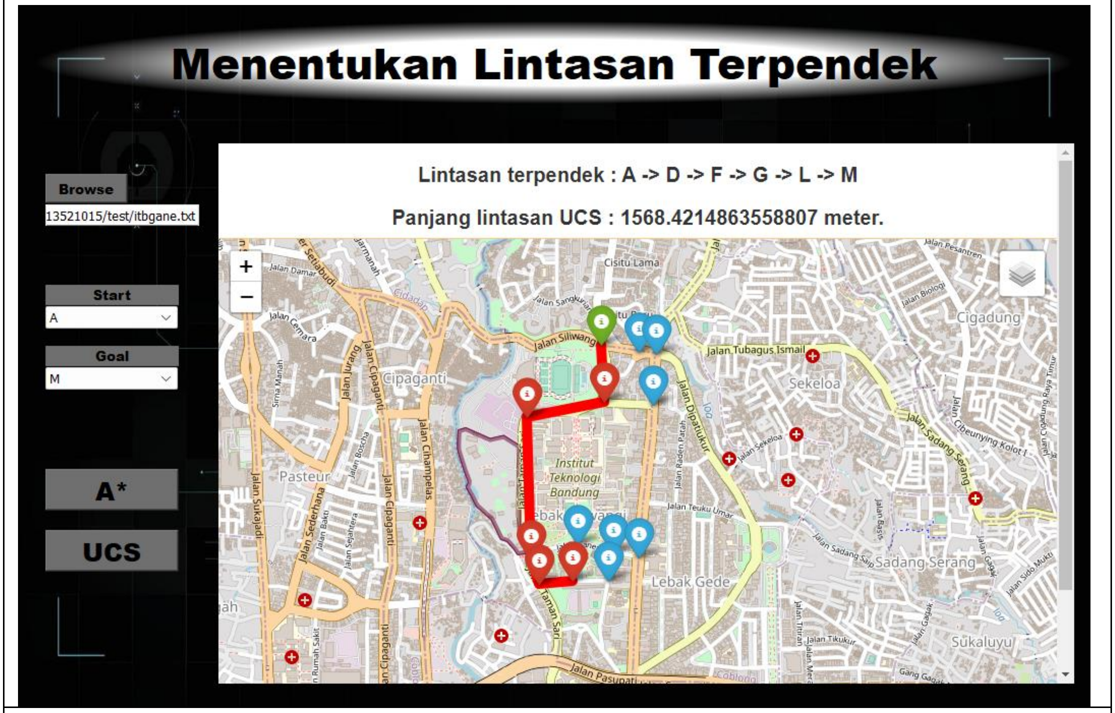
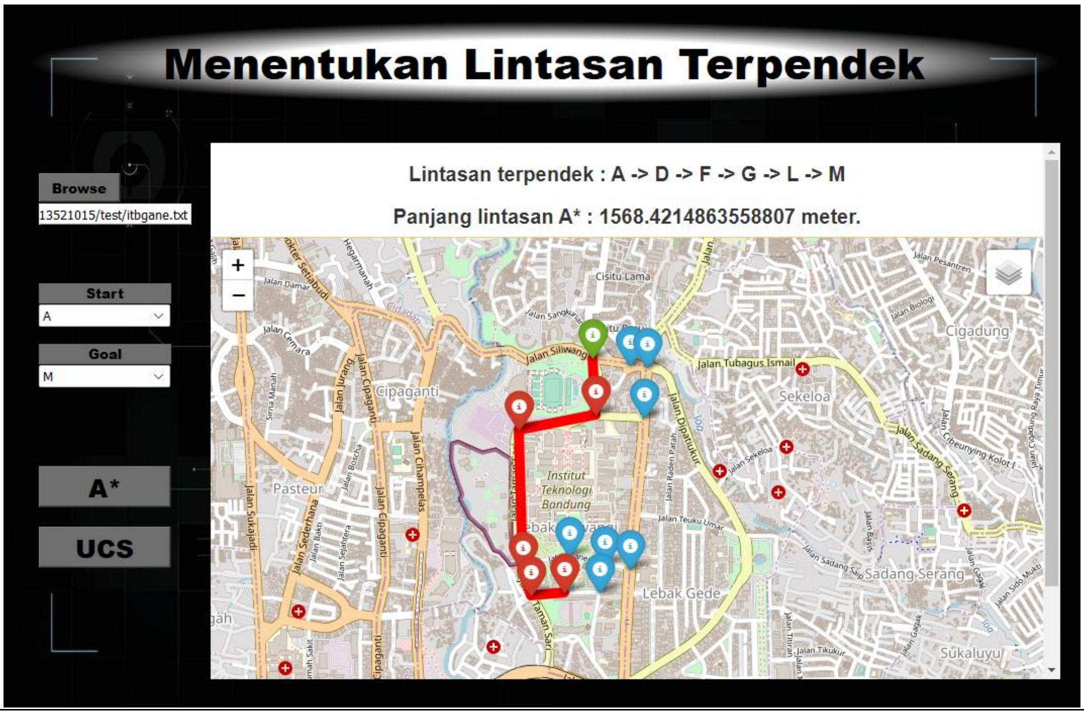
   Node start = A

   Node goal = M



```
13
A -6.884893 107.611440
B -6.885191 107.613010
C -6.885257 107.613730
D -6.887256 107.611540
E -6.887386 107.613610
F -6.887910 107.608280
G -6.893882 107.608450
H -6.893230 107.610440
I -6.893605 107.611940
J -6.893780 107.613030
K -6.894759 107.611720
L -6.894883 107.608830
M -6.894775 107.610230
0 1 0 1 0 0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0 0 0
1 1 0 0 1 1 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0 1 0
0 0 0 0 0 0 1 0 1 0 0 0 1
0 0 0 0 0 0 0 1 0 1 1 0 0
0 0 0 0 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 0 0 0 1 0 1
0 0 0 0 0 0 0 1 0 0 0 1 0
```

Output:

UCS

A*


UCS

Menentukan Lintasan Terpendek

13521015/test/itbgane.txt

Start: A

Goal: M

Lintasan terpendek : A -> D -> F -> G -> L -> M

Panjang lintasan UCS : 1568.4214863558807 meter.

A*

Menentukan Lintasan Terpendek

13521015/test/itbgane.txt

Start: A

Goal: M

Lintasan terpendek : A -> D -> F -> G -> L -> M

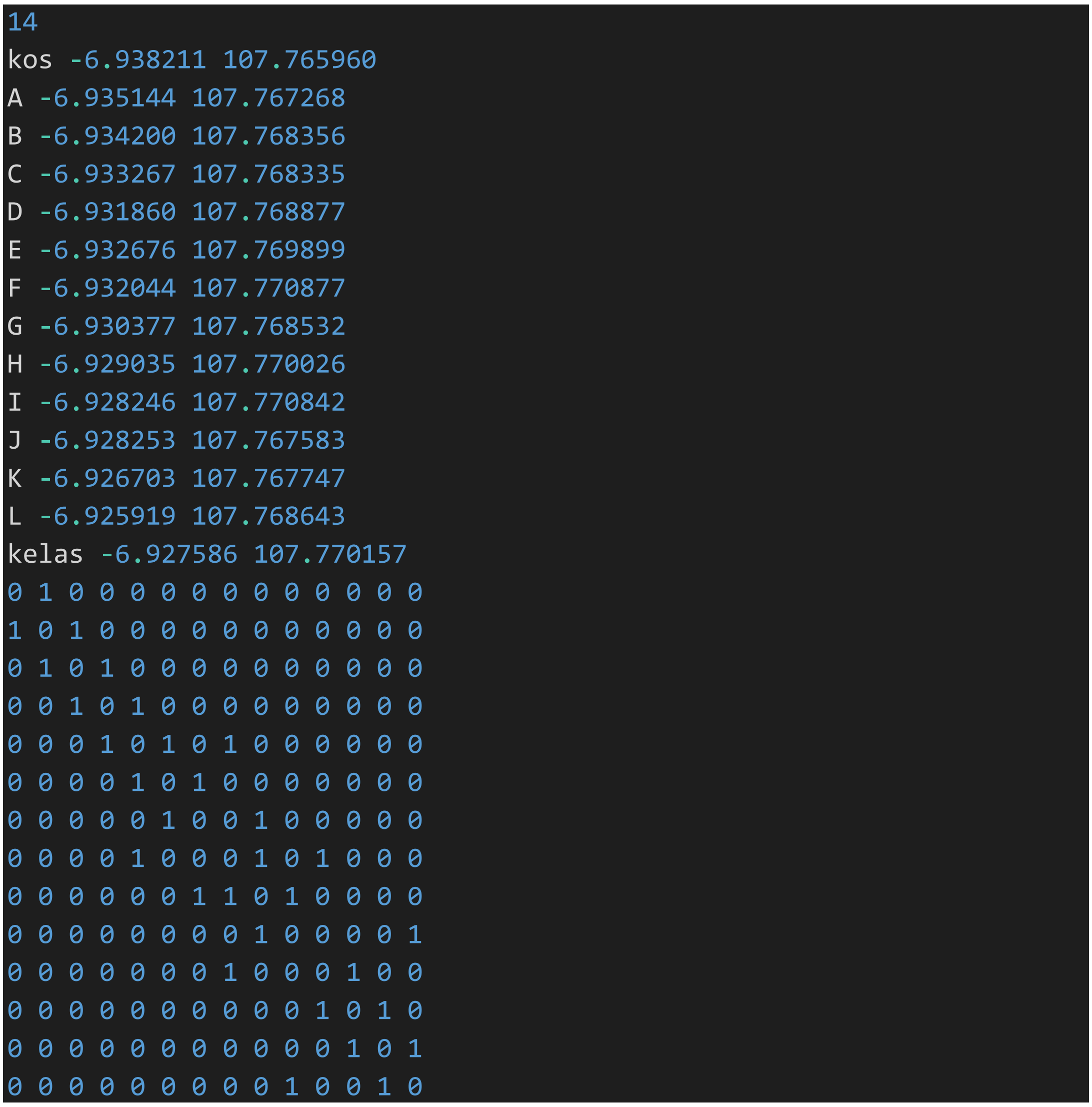
Panjang lintasan A* : 1568.4214863558807 meter.

5. itbnangor.txt

   Input:

   Node start = kos

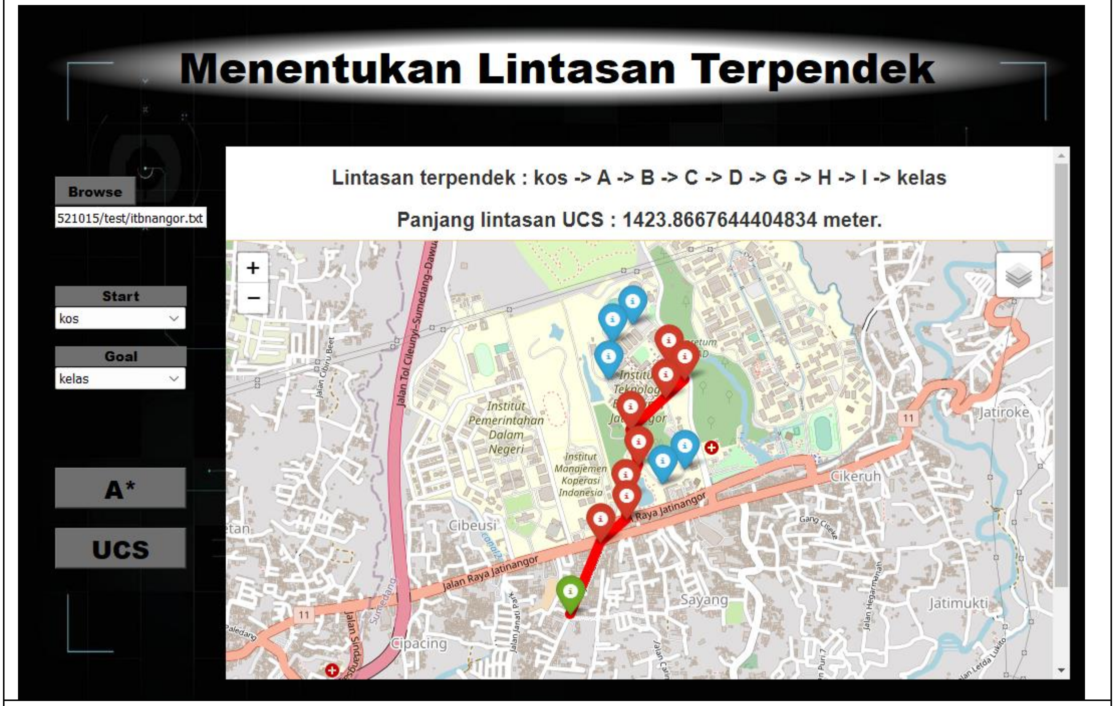
   Node goal = kelas



```
14
kos -6.938211 107.765960
A -6.935144 107.767268
B -6.934200 107.768356
C -6.933267 107.768335
D -6.931860 107.768877
E -6.932676 107.769899
F -6.932044 107.770877
G -6.930377 107.768532
H -6.929035 107.770026
I -6.928246 107.770842
J -6.928253 107.767583
K -6.926703 107.767747
L -6.925919 107.768643
kelas -6.927586 107.770157
0 1 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 1 0 1 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 1 0 1 0 1 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 1 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 1 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 1 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 1 0 1
0 0 0 0 0 0 0 0 0 1 0 0 1 0
```

Output:

UCS



Menentukan Lintasan Terpendek

Browse 521015/test/itbnangor.txt

Start kos

Goal kelas

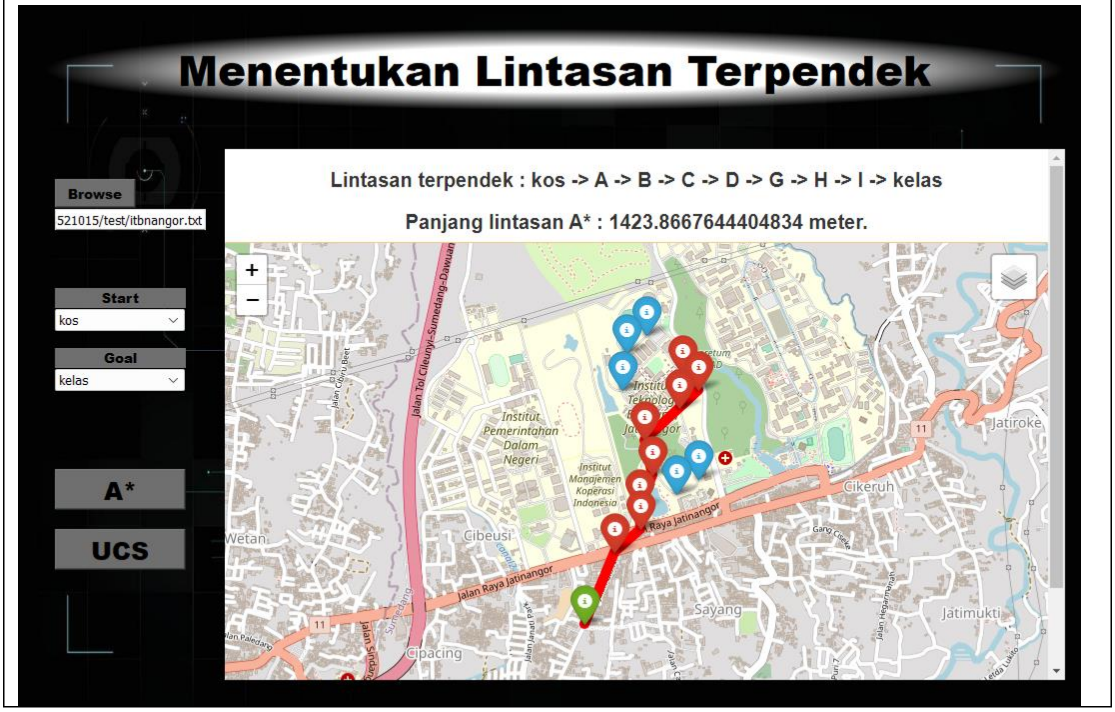
A*

UCS

Lintasan terpendek : kos -> A -> B -> C -> D -> G -> H -> I -> kelas

Panjang lintasan UCS : 1423.8667644404834 meter.

A*



Menentukan Lintasan Terpendek

Browse 521015/test/itbnangor.txt

Start kos

Goal kelas

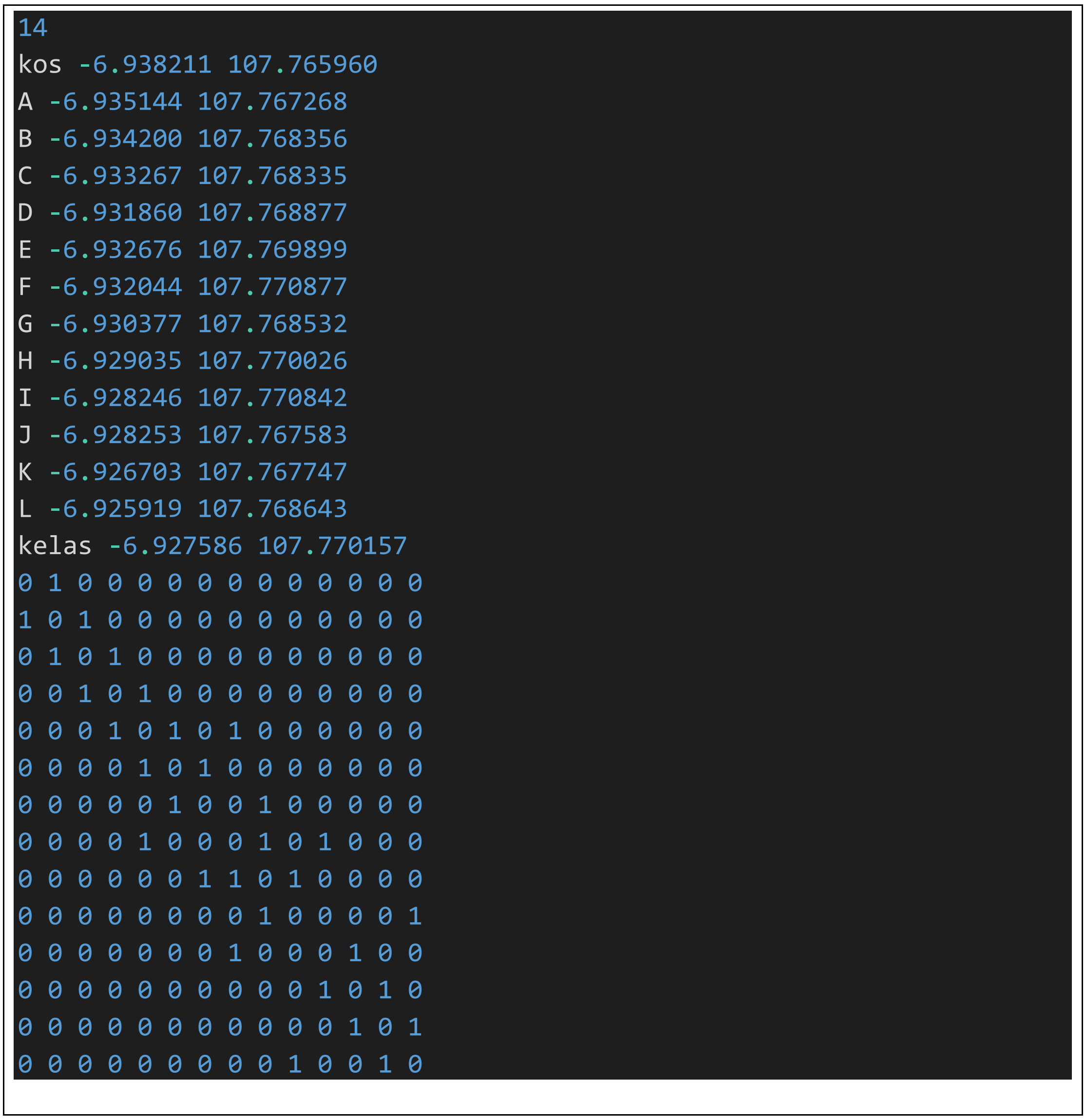
A*

UCS

Lintasan terpendek : kos -> A -> B -> C -> D -> G -> H -> I -> kelas

Panjang lintasan A* : 1423.8667644404834 meter.

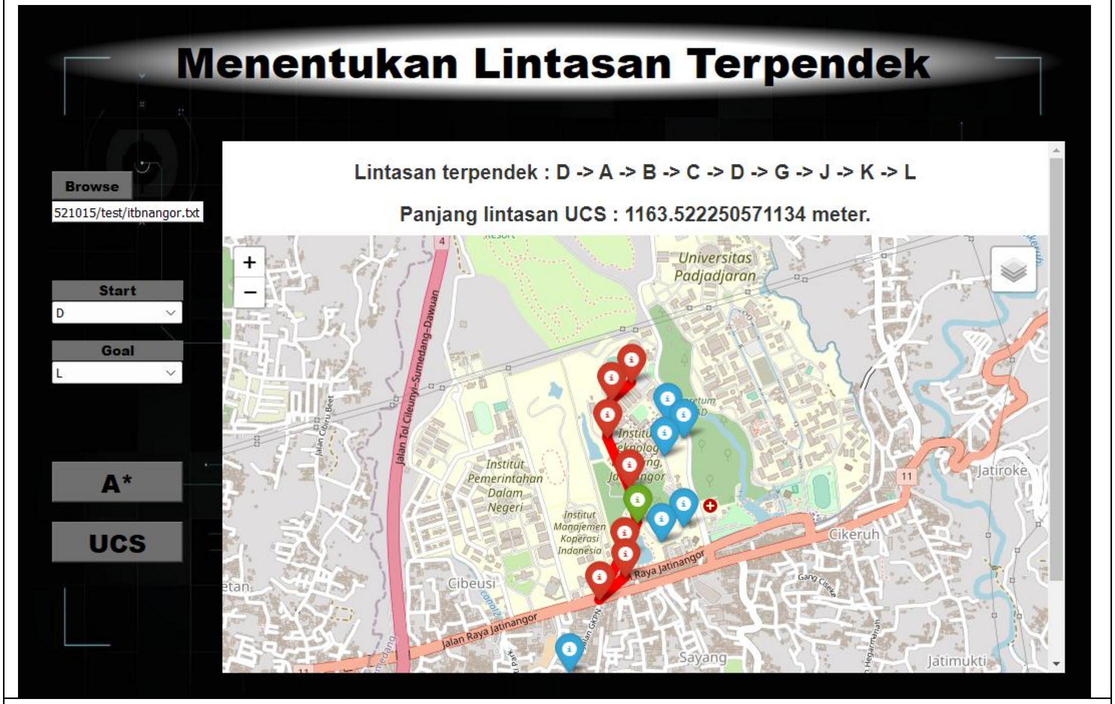
6. itbnangor.txt
   Input:
   Node start = D
   Node goal = L



```
14
kos -6.938211 107.765960
A -6.935144 107.767268
B -6.934200 107.768356
C -6.933267 107.768335
D -6.931860 107.768877
E -6.932676 107.769899
F -6.932044 107.770877
G -6.930377 107.768532
H -6.929035 107.770026
I -6.928246 107.770842
J -6.928253 107.767583
K -6.926703 107.767747
L -6.925919 107.768643
kelas -6.927586 107.770157
0 1 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 1 0 1 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 1 0 1 0 1 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 1 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 1 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 1 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 1 0 1
0 0 0 0 0 0 0 0 0 1 0 0 1 0
```

Output:

UCS



Menentukan Lintasan Terpendek

Lintasan terpendek : D -> A -> B -> C -> D -> G -> J -> K -> L

Panjang lintasan UCS : 1163.522250571134 meter.

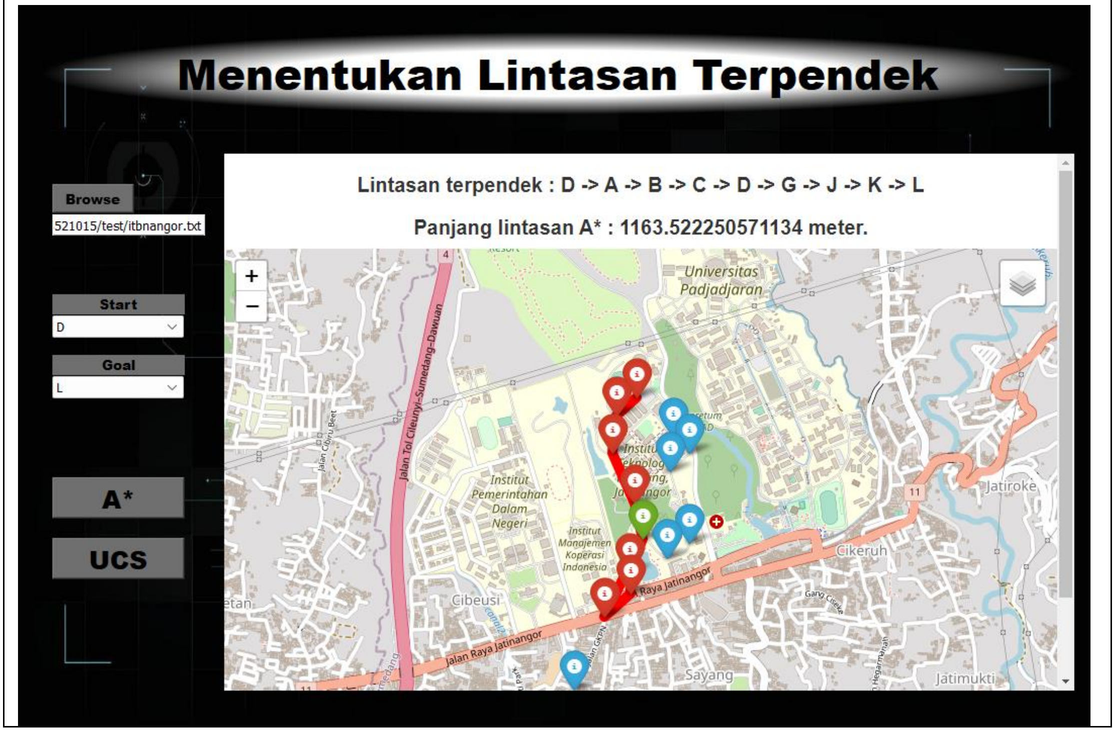
Start: D

Goal: L

521015/test/itbnangor.txt

Browse
A*
UCS
Universitas Padjadjaran
Institut Pemerintahan Dalam Negeri
Institut Manajemen Koperasi Indonesia
Cibeusi
Cikeruh
Jatiroke
Sayang
Jatimukti
Jalan Raya Jatinangor
Jalan Tol Cileunyi-Sumedang-Dawuan

A*



Menentukan Lintasan Terpendek

Lintasan terpendek : D -> A -> B -> C -> D -> G -> J -> K -> L

Panjang lintasan A* : 1163.522250571134 meter.

Start: D

Goal: L

521015/test/itbnangor.txt

Browse
A*
UCS
Universitas Padjadjaran
Institut Pemerintahan Dalam Negeri
Institut Manajemen Koperasi Indonesia
Cibeusi
Cikeruh
Jatiroke
Sayang
Jatimukti
Jalan Raya Jatinangor
Jalan Tol Cileunyi-Sumedang-Dawuan

BAB VI
PRANALA GITHUB

https://github.com/WildanGhaly/Tucil3_13521015

BAB VII

KESIMPULAN

Kesimpulan:

Dari pembuatan tugas kecil 3 strategi algoritma penulis menyimpulkan jika algoritma A* dan UCS dapat mencari jarak terdekat dari titik awal dan titik akhir yang dipilih pada sebuah graf. Eksekusi waktu program bergantung kepada kasus-kasus tertentu.

Komentar:

Saya berharap kedepannya spesifikasi tugas dapat diperjelas lagi

BAB VIII

LAMPIRAN

No	Keterangan	Checklist
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	√