

IMPLEMENTASI ALGORITMA A* UNTUK MENENTUKAN LINTASAN TERPENDEK

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II 2020/2021

Disusun oleh

Gde Anantha Priharsena (13519026)

Reihan Andhika Putra (13519043)



**TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021**

BAB I

SOURCE CODE PROGRAM

1. File Astar.py

```
# Import library
import math

# Import kodingan sendiri
from Haversine import haversine_distance

# =====Class Definition=====
"""
class Astar_Node merepresentasikan node pada state space tree yang dibentuk saat algoritma
pencarian astar dilakukan. Astar_Node nantinya akan dimasukkan ke dalam antrian dimana Astar_Node
dengan total harga terendah dan belum dikunjungi akan dikunjungi terlebih dahulu. Astar_Node yang
sudah dikunjungi akan disimpan ke dalam list.
"""
class Astar_Node:
    """
    path adalah node yang dilalui untuk mencapai node saat ini
    g adalah harga dari node start hingga node saat ini dengan path tertentu
    h adalah estimasi harga dari node saat ini ke node tujuan secara heuristik
    f adalah total harga dari node saat ini (f = g + h)
    """
    def __init__(self, path=None, index=None, g=0, h=0, f=0):
        # Constructor
        # Mengisi attribute kelas astar dengan attribute default atau attribute masukan

        self.path = path
        self.index = index

        self.g = g
        self.h = h
        self.f = f

    def __eq__(self, other):
        # Operator overloading ==
        # Dua buah Astar_Node dikatakan sama apabila indexnya sama
        return self.index == other.index

    def __str__(self):
        # Function overloading str()
        # Mengubah representasi Astar_Node saat di print
        return(self.path+ " f: "+ str(self.f) +" g: "+ str(self.g)+" h: "+str(self.h))
```

```

# Algoritma pencarian dengan astar
def astar_find(graph,start,end):
    # I.S graph adalah graf yang valid, start dan end adalah node yang valid
    # F.S Mengirimkan path dengan jalur terpendek dari node start ke node end

    # Inisialisasi Astar_Node dari node start dan node goal
    start_astar_node = Astar_Node(path=str(start.index)+"-",index=start.index)
    end_astar_node = Astar_Node(path="",index=end.index)

    # Deklarasi list node yang sudah dikunjungi dan akan dikunjungi
    to_visit = []
    has_visited = []

    # Astar_Node pertama adalah Astar_Node start
    to_visit.append(start_astar_node)

    # Selama masih ada Astar_Node yang bisa dikunjungi lakukan algoritma pencarian astar
    while (len(to_visit) > 0):

        # Astar_Node yang akan dikunjungi diurutkan berdasarkan nilai f nya
        to_visit.sort(key= lambda x:x.f)

        # Ambil Astar_Node dengan nilai f terkecil dan kunjungi Astar_Node tersebut, simpan sebagai
        # curr_astar_node
        curr_astar_node = to_visit[0]
        to_visit.pop(0)
        has_visited.append(curr_astar_node)

        # Jika curr_astar_node yang sekarang merupakan node tujuan maka return pathnya
        if (curr_astar_node==end_astar_node):
            return [curr_astar_node.path[:len(curr_astar_node.path)-1], curr_astar_node.g]

        # Inisialisasi calon Astar_Node yang akan dikunjungi
        astar_candidate = []

        """
        1) Untuk semua tetangga dari curr_astar_node yang ada jadikan dia calon Astar_Node yang
        akan dikunjungi
        2) Untuk semua tetangga dari curr_astar_node
            1> path nya adalah path dari curr_astar_node ditambah index nya
            2> h nya adalah haversian_distance nya terhadap node goal
            3> g nya harga dari curr_astar_node sekarang ditambah harga menuju tetangga
            4> f nya adalah h + g
        """
        for i in range(graph.size):
            if(graph.adj[curr_astar_node.index][i] != math.inf):
                heurisitc_price = haversine_distance(graph.find_node(index=i),graph.find_node
                    (index=end_astar_node.index))
                path_price = curr_astar_node.g + graph.adj[curr_astar_node.index][i]
                new_astar_node = Astar_Node(path=curr_astar_node.path+str(i)+"-", index=i, g=path_price
                    , h=heurisitc_price, f=heurisitc_price+path_price)
                astar_candidate.append(new_astar_node)

        # Evaluasi apakah calon Astar_Node layak untuk dikunjungi
        for astar_node in astar_candidate:

            # Jika ditemukan Astar_Node yang sama (==) di list sudah dikunjungi maka jangan kunjungi
            # Astar_Node kandidat
            if(len([visited_astar_node for visited_astar_node in has_visited if visited_astar_node ==
                astar_node])>0):
                continue

            # Jika ditemukan Astar_Node yang sama (==) dan berada di list akan dikunjungi dan
            # Astar_Node yang sekarang
            # F nya lebih mahal (Astar_Node yang sekarang bukan best solution so far) maka jangan
            # kunjungi Astar_Node kandidat
            if (len([astar_node_2 for astar_node_2 in to_visit if astar_node == astar_node_2 and
                astar_node.f > astar_node_2.f])) > 0:
                continue

            # Astar_Node sekarang merupakan calon the best solution so far
            to_visit.append(astar_node)

        # Tidak ditemukan path
        return [None,"infinity"]

# =====

```

2. File Graph.py

```
# Import Library
import math

# =====Class
Definition=====
"""
    Class Node merepresentasikan persimpangan pada suatu jalan. Node disini berbeda dengan Astar_Node
    dimana Astar_Node adalah node yang digunakan untuk membantu pencarian dengan algoritma astar. Node
    disini murni representasi dari suatu simpangan
"""
class Node:
    """
        index adalah index node pada adjacency matriks di graf
        name adalah nama persimpangan
        longitude adalah garis bujur(dalam derajat) dari persimpangan
        latitude adalah garis lintang(dalam derajat) dari persimpangan
    """
    def __init__(self, name="", index=0, longitude=0, latitude=0):
        # Constructor
        # Mengisi attribute kelas Node dengan atribut default atau atribut masukan
        self.index = index
        self.name = name
        self.longitude = longitude
        self.latitude = latitude

    def __str__(self):
        # Function overloading str()
        # Mengubah representasi Node saat di print
        return (str(self.index)+ " "+self.name+" "+str(self.longitude)+ " "+str(self.latitude))

"""
    Class Graph merupakan representasi Graf dengan sisi yaitu jalan antar persimpangan
    Graf direpresentasikan dengan matriks ketetanggaan berbobot. Graf akan di-load dari file
    external dan jarak antar node yang bertetangga akan dihitung dengan haversian_distance
"""

class Graph():
    """
        size adalah banyaknya node sekaligus menjadi ukuran matriks ketetanggaan (size*size)
        list_of_node merupakan list yang berisikan informasi umum dari node yang ada di graf
        adj merupakan matriks ketetanggaan dari graf
    """
    def __init__(self, size=0, adj=[], list_of_node = []):
        # Constructor
        # Mengisi attribute kelas Graph dengan atribut default atau atribut masukan

        self.size = size

        self.list_of_node = []
        for node in list_of_node:
            self.list_of_node.append(node)
        self.adj = []
```



```
if (adj==[]):
    for i in range(size):
        self.adj.append([math.inf for i in range(size)])
else:
    for row in adj:
        adj_row = []
        for col in row:
            adj_row.append(col)
        self.adj.append(adj_row)

def find_node(self,index=None,name=None):
    # Menemukan node dengan parameter index atau nama
    try:
        if(index!= None ):
            return [node for node in self.list_of_node if node.index==index][0]
        if(name!= None ):
            return[node for node in self.list_of_node if node.name==name][0]
    except:
        print("invalid index or name")

def add_edge(self, orig, dest, length = 0):
    # Menambahkan sisi antar node dengan memodifikasi matriks ketetanggaan
    try:
        if (orig == dest): raise IndexError
        self.adj[orig][dest] = length
        self.adj[dest][orig] = length
    except IndexError:
        print("Invalid index")

def remove_edge(self, orig, dest):
    # Menghilangkan sisi antar node dengan memodifikasi matriks ketetanggaan
    try:
        if (orig == dest): raise IndexError
        self.adj[orig][dest] = math.inf
        self.adj[dest][orig] = math.inf
    except IndexError:
        print("Invalid index")

def display_adj(self):
    # Menampilkan matriks ketetanggaan
    for row in self.adj:
        for val in row:
            print(val,end=(8-len(str(val)))*" ")
        print()

def transform_path(self,path):
    # Merubah path dari index dengan nama
    new_path = ""
    path= path.split('-')
    for node in path:
        new_path= new_path+ self.find_node(index=int(node)).name + "-"
    return new_path[:len(new_path)-1]

# =====
```

3. File Haversine.py

```
# Import library
import math

# Import kodingan sendiri
from Graph import Node

# =====Function=====
# Menghitung jarak antara dua node dengan latitude dan longitude nya menggunakan formula haversine
def haversine_distance(node1, node2):
    # I.S node 1 dan node 2 adalah objek class Node yang valid
    # F.S Mengirimkan jarak antara node1 dan node2 dengan satuan meter dibulatkan 3 angka di belakang
    # koma

    # longitude dan latitude dalam derajat
    lon_1 = node1.longitude
    lat_1 = node1.latitude
    lon_2 = node2.longitude
    lat_2 = node2.latitude

    # R = radius bumi dalam satuan meter
    R = 6371000

    # Mengubah latitude dan delta latitude dan longitude dalam radian
    lat_1_rad = math.radians(lat_1)
    lat_2_rad = math.radians(lat_2)
    delta_lat = math.radians(lat_2 - lat_1)
    delta_lon = math.radians(lon_2 - lon_1)

    # Formula haversine
    a = math.sin(delta_lat / 2.0) ** 2 + math.cos(lat_1_rad) * math.cos(lat_2_rad)
        * math.sin(delta_lon / 2.0) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = R*c

    # Bulatkan angka dibelakang koma
    return round(d, 3)

# =====
```

4. File Parser.py

```
# Import Library
import math

# Import kodingan sendiri
from Graph import Node, Graph
from Haversine import haversine_distance

# =====Class
Definition=====
"""
    class Parser adalah class yang digunakan untuk mengekstrak input dari file external
    menjadi graf
"""
class Parser:
    def __init__(self):
        # Constructor
        # Mengisi attribute kelas astar dengan atribut default atau atribut masukan
        self.graph = Graph()
        self.bool_adj = []
        self.list_of_node = []
        self.num_of_node = 0

    def read_from_file(self, filename):
        inputs = open('test/'+ filename +'.txt', 'r').read().split('\n')

        # Dapatkan num_of_node pada input txt
        self.num_of_node = int(inputs[0])

        # Dapatkan informasi node pada input txt
        for i in range(1, 1+self.num_of_node):
            input = inputs[i].split(' ')
            self.list_of_node.append(Node(name=input[0], index=i-1,
            latitude=float(input[1]), longitude=float(input[2])))

        # Dapatkan adjacency matrix dalam bentuk boolean pada input txt
        for i in range(self.num_of_node+1, 2*self.num_of_node+1):
            bool_adj_row = []
            input = inputs[i].split(' ')
            for is_adj in input:
                bool_adj_row.append(int(is_adj))
            self.bool_adj.append(bool_adj_row)

        # Buat graf permulaan dan isi adjacency matrixnya satu per satu berdasarkan adjacency matrix
        boolean
        self.graph = Graph(size=self.num_of_node, list_of_node=self.list_of_node)
        for i in range(self.num_of_node):
            for j in range(self.num_of_node):
                if (self.bool_adj[i][j] == 1 and self.graph.adj[i][j] == math.inf):
                    self.graph.add_edge(i, j, haversine_distance(self.list_of_node[i], self.list_of_node[j]))

    def display_attr(self, bool_adj=False, node=False, graph_adj=False):
        # Menampilkan data dari file txt dan menampilkan adjacency matriks pada graf
        if bool_adj:
            print("Boolean Adjacency matrix")
            for row in self.bool_adj:
                for col in row:
                    print(col, end=(3-len(str(col)))*" ")
                print()
        if graph_adj:
            print()
            print("Graph weighted adjacency matrix")
            self.graph.display_adj()
        if node:
            print()
            print("List of node")
            for node in self.list_of_node:
                print(node)

# =====
```

5. File main.py

```
# Main Program tanpa visualisasi untuk mengecek jalur terpendek
from Graph import Graph
from Astar import astar_find
from Parser import Parser

import os
os.chdir("../") # Pindah ke directory atas

# Membaca input dari file txt dan menjadikannya dalam bentuk graf
filename = input("Masukkan nama file (tanpa ekstensi): ")
filecontent = Parser()
filecontent.read_from_file(filename)

# Menuliskan data yang ada pada file txt dan graf yang dibentuk dari file txt
filecontent.display_attr(node=True, bool_adj=True, graph_adj=True)

# Meload graf dari file txt
graph = filecontent.graph

# Masukkan nama simpul awal dan simpul tujuan
start_node_name = input("Masukkan nama node awal: ")
end_node_name = input("Masukkan nama node tujuan: ")

# Cari simpul dengan nama tersebut di dalam graf
start_node = graph.find_node(name=start_node_name)
end_node = graph.find_node(name=end_node_name)

# Melakukan pencarian dengan algoritma astar
path = astar_find(graph, start_node, end_node)[0]
price = astar_find(graph, start_node, end_node)[1]

# Menuliskan jalur yang didapat
print(graph.transform_path(path))
print(price)
```


6. File main.ipnyb

Section 1 - Import Library dan Kode Yang Sudah Dibuat

Library yang digunakan untuk memvisualisasi peta adalah folium. Library Numpy, pandas, dan math digunakan untuk membantu perhitungan. Kode yang sudah dibuat terletak pada folder src(sama seperti notebook ini). File kode yang digunakan di notebook ini adalah Astar.py, Graph.py, Haversine.py, dan Parser.py. Library os digunakan supaya notebook ini bisa membaca file dari direktori yang berbeda.

```
# Import Library yang dibutuhkan untuk visualisasi
import folium
from folium import plugins
import numpy as np
import pandas as pd

# Import kode yang sudah dibuat
import Astar
import Graph
import Haversine
import Parser
import math

# Import fungsi spesifik
from Graph import Graph
from Astar import astar_find
from Parser import Parser

# Import library os untuk pindah direktori supaya bisa mendapatkan file testcase
import os
os.chdir("../") # Pindah ke direktori atas
```

Section 2 - Baca File dan Visualisasi Peta Awal

Melakukan pembacaan file eksternal dan memvisualisasikan graf beserta peta dari file eksternal tersebut. Tahap-tahap yang harus dilalui adalah sebagai berikut

1. Masukkan nama file eksternal dengan benar
2. Apabila ditemukan error karena nama file yang dimasukkan salah, run kembali cellnya dan masukkan nama file hingga benar
3. Program akan menampilkan peta beserta graf dari file eksternal
4. Apabila peta atau graf kurang jelas, cobalah zoom-in atau zoom-out hingga peta dan graf enak untuk dilihat
5. Tekanlah marker yang ada di peta untuk melihat nama dari simpul yang ada di graf

```
# Membaca input dari file txt dan menjadikannya dalam bentuk graf
print("Nama File yang tersedia :")
print("1.ITB")
print("2.Alun-Alun")
print("3.Buah-Batu")
print("4.Bojonegoro")
```

```

print("5.Bogor")
print("6.Jakarta")
filename = input("Masukkan nama file (tanpa ekstensi): ")
filecontent = Parser()
try:
    filecontent.read_from_file(filename)
    graph = filecontent.graph
except:
    print("Tidak ada file dengan ekstensi tersebut")

# Membentuk rute awal pada peta yang akan digambar
list_of_adj = graph.adj
list_of_route = []
for i in range(len(list_of_adj)):
    for j in range(len(list_of_adj[i])):
        tempRoute=[[graph.find_node(index=int(i)).latitude,graph.find_node(index
                        =int(i)).longitude]]
        if(list_of_adj[i][j]!=math.inf):
            tempRoute.append([graph.find_node(index=int(j)).latitude,graph.find_
                                node(index=int(j)).longitude])
            list_of_route.append(tempRoute)

list_of_node = graph.list_of_node

# create map
map_cities = folium.Map(location=[list_of_node[5].latitude, list_of_node[5].long
itide], zoom_start=15, control_scale=True)

# plot locations
for node in list_of_node :
    folium.Marker(location=[node.latitude, node.longitude], popup=node.name).add
        _to(map_cities)

# plot route for each marker
for route in list_of_route:
    folium.PolyLine(route, color="blue", weight=1).add_to(map_cities)

# display map
map_cities

```

Section 3 - Membaca Simpul Awal, Tujuan dan Menampilkan Lintasan Terpendek

Melakukan pembacaan simpul awal dan tujuan kemudian menampilkan lintasan terpendek dari node awal ke tujuan dengan menggunakan algoritma astar. Tahap-tahap yang harus dilalui adalah sebagai berikut

1. Masukkan nama simpul awal dan tujuan dengan benar, nama simpul bisa dilihat dengan menekan marker pada peta
2. Apabila ditemukan error karena nama simpul yang dimasukkan salah, run kembali cellnya dan masukkan nama simpul hingga benar

3. Program akan menampilkan peta beserta lintasan terpendeknya dengan jalur yang berwarna merah

4. Diatas peta, program juga akan menampilkan panjangnya jalan yang ditempuh

```
# Masukkan nama simpul awal dan simpul tujuan
start_node_name = input("Masukkan nama simpul awal: ")
end_node_name = input("Masukkan nama simpul tujuan: ")

# Cari simpul dengan nama tersebut di dalam graf
try:
    start_node = graph.find_node(name=start_node_name)
    end_node = graph.find_node(name=end_node_name)
except:
    print("Tidak ada node dengan nama tersebut")

# Melakukan pencarian dengan algoritma astar
result = astar_find(graph,start_node,end_node)
path = result[0]
price = result[1]

# Jika tidak ditemukan jalur
if (path==None):
    print("Tidak ada jalur ditemukan")

# Bentuk rute dari path yang didapat
path = path.split('-')
list_of_node = []
route = []
for x in path:
    list_of_node.append(graph.find_node(index=int(x)))
for x in list_of_node:
    route.append([x.latitude, x.longitude])

# tambahkan rute ke peta
plugins.AntPath(route, color="red", weight=8).add_to(map_cities)

# tampilkan harga
print("Panjang perjalanan yang harus ditempuh adalah "+str(price)+" m")
# tampilkan peta
map_cities
```

Section 4 - Lampiran

Akan ditampilkan booleand adjacency matrix (dari file .txt), info node(dari graf yang dibuat), dan weighted adjacency matrix(dari graf). Bagian ini bisa digunakan untuk mengkoreksi apakah A* yang dibuat sudah menghasilkan jalur terpendek atau tidak.

Funfact : Seringkali jika dilihat dengan mata, jalur yang terpilih bukanlah jalur terpendek namun saat dihitung ternyata selisihnya dengan jalur lain cuma 1-2 meter saja :D jadi silahkan dikoreksi dengan perhitungan

```
filecontent.display_attr(node=True,bool_adj=True,graph_adj=True)
```

Melakukan Pencarian Lagi

Apabila ingin melakukan pencarian lagi, ada beberapa cara yang bisa dilakukan.

Cara 1

Pilih "Kernel" pada toolbar di atas Jupyter Notebook. Setelah itu pilih "Restart & Clear Output" untuk merestart notebook. Setelah itu anda bisa run cell nya satu per-satu lagi dari awal.

Cara 2

Pilih "Kernel" pada toolbar di atas Jupyter Notebook. Setelah itu pilih "Restart & Run All" untuk merestart notebook dan run semua cell nya satu persatu. Apabila menggunakan cara ini pastikan input selalu benar di setiap tahap.

Cara 3

Run cell nya satu persatu mulai dari Section 2 agar input peta bisa terefresh, jangan mulai dari section 1. Apabila ada hal aneh yang terjadi maka gunakan cara 1 atau cara 2 saja.

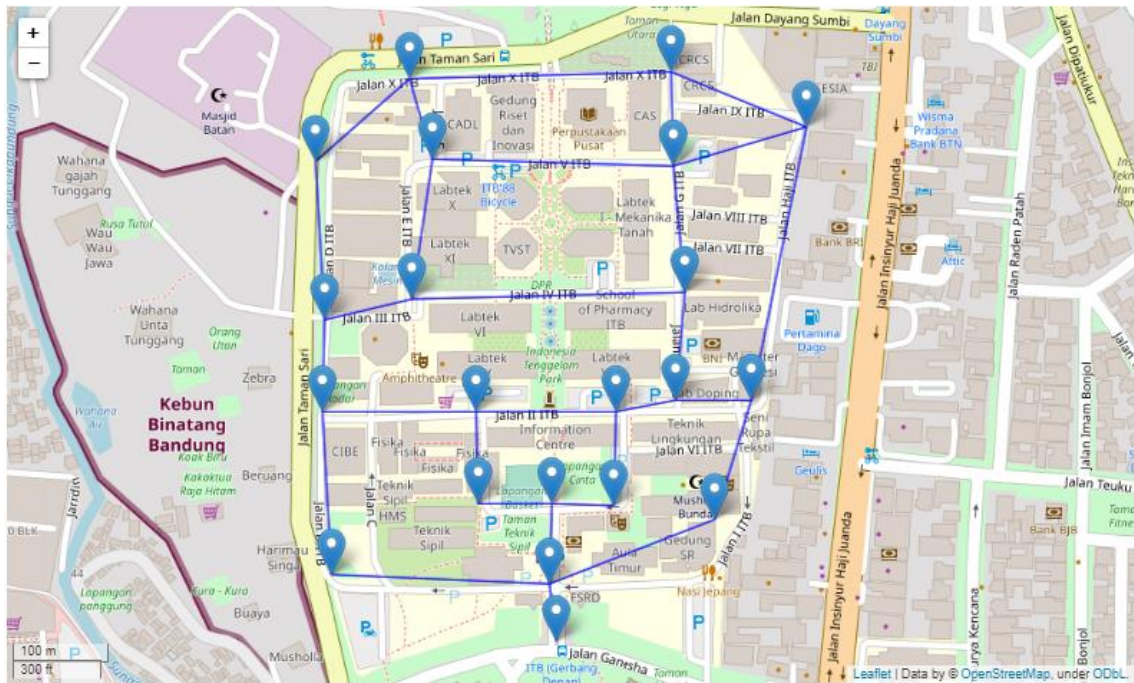
BAB II

Hasil Pengujian

File yang digunakan untuk pengujian, yaitu:

1. ITB.txt

```
21
n_0 -6.893152249630082 107.61043379113852
n_1 -6.89261968510423 107.61036941812557
n_2 -6.892534474724565 107.60833093938244
n_3 -6.89202321212487 107.61191437043615
n_4 -6.891895396388793 107.61097023291303
n_5 -6.891874093762762 107.61039087579655
n_6 -6.891874093762762 107.60970423032518
n_7 -6.891021987936512 107.6096827726542
n_8 -6.891021987936512 107.61099169058402
n_9 -6.8909154746005585 107.61154959002951
n_10 -6.8909154746005585 107.61225769317184
n_11 -6.891021987936512 107.60824510869853
n_12 -6.890169880578749 107.6082665663695
n_13 -6.889978156212211 107.60908195786675
n_14 -6.889914248072802 107.61163542071343
n_15 -6.888678689017943 107.60927507690558
n_16 -6.8886999917875436 107.60818073568558
n_17 -6.888742597323873 107.61152813235852
n_18 -6.888380450143099 107.61277267727537
n_19 -6.887869183064192 107.61150667468755
n_20 -6.887935272105543 107.60905981049973
01000000000000000000000000000000
10110100000000000000000000000000
01000000000010000000000000000000
01000000000010000000000000000000
00000100100000000000000000000000
01001010000000000000000000000000
00000101000000000000000000000000
00000010100100000000000000000000
00001001010000000000000000000000
00000000101000100000000000000000
00010000010000000000000000000000
00100001000010000000000000000000
00000000000010100100000000000000
00000000000000101100000000000000
000000000100010001000100010001000
000000000000001000100100010001001
00000000000000100000000100000001
00000000000000001100110
000000000010000001010
00000000000000000001101
00000000000000000110010
```



Gambar 1. Peta ITB

2. Alun-Alun.txt

```

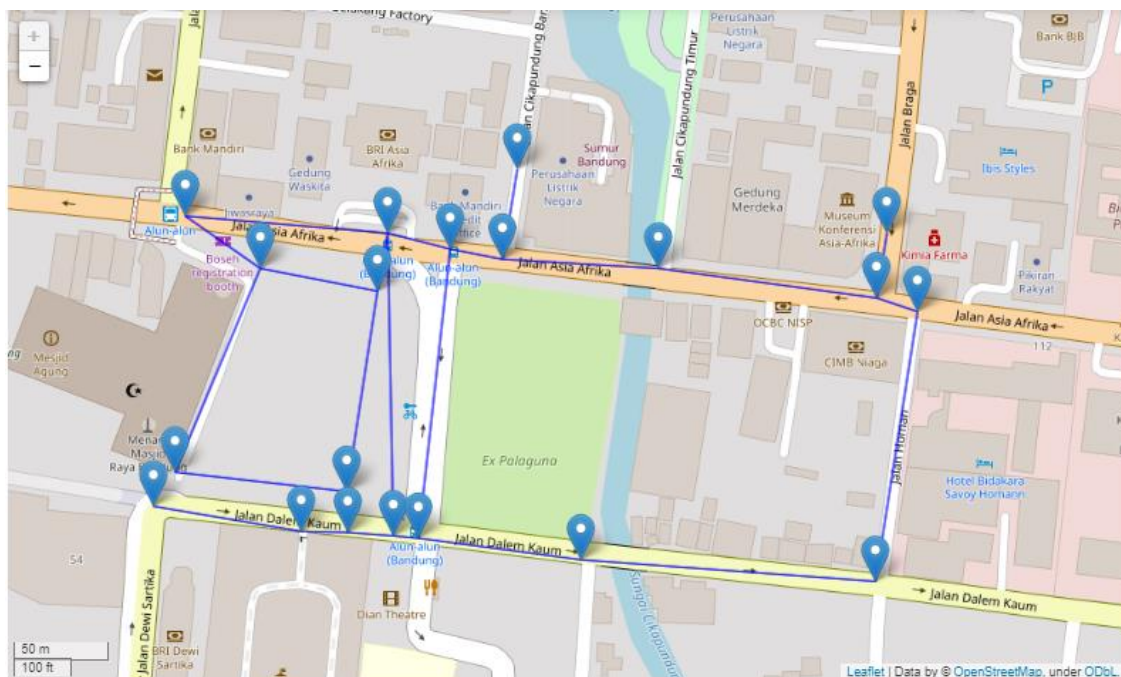
20
n_0 -6.921060932511885 107.60653401156785
n_1 -6.921305897537567 107.60688806316733
n_2 -6.921412404030811 107.60744059823928
n_3 -6.922259129797078 107.60648573180427
n_4 -6.92234966013503 107.60729575894858
n_5 -6.921140812425525 107.60748887800287
n_6 -6.922541371381588 107.60730112336675
n_7 -6.922557347319368 107.60751570009187
n_8 -6.921215367003489 107.60778392100059
n_9 -6.922567997941536 107.6076337172917
n_10 -6.9212632949328325 107.6080306842067
n_11 -6.921305897536867 107.60876560949657
n_12 -6.9214443559741925 107.60979557776281
n_13 -6.921508259854394 107.60998869681119
n_14 -6.922770359713114 107.60979021333873
n_15 -6.92266917884569 107.60840082904143
n_16 -6.921140812430106 107.60984385751702
n_17 -6.922418889229562 107.6063838078644
n_18 -6.920842594008965 107.60810042164739
n_19 -6.922536046082426 107.60708118222088
0 1 0 0 0 1 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 1 0 0 1 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 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 1 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 1
0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0

```

```

00000100011000000000
00000001100000010000
00000000100100000010
00000000001010000000
00000000000101001000
00000000000010100000
00000000000001010000
00000000000000100000
000000000010000100000
00000000000001000000
000000000000000000001
000000000001000000000
000000100000000000100

```



Gambar 2. Peta Alun-Alun Bandung

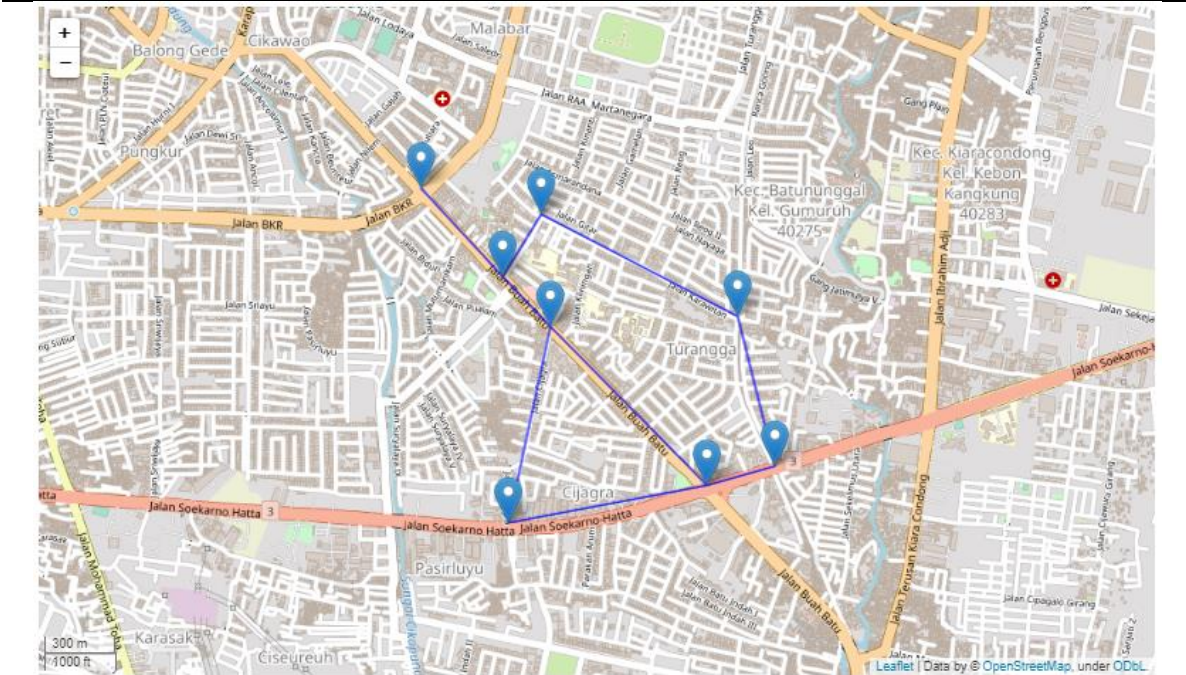
3. Buah-Batu.txt

```

8
n_0 -6.936757 107.622691
n_1 -6.947901 107.633486
n_2 -6.947165 107.636042
n_3 -6.941532 107.634648
n_4 -6.937735 107.627215
n_5 -6.940052 107.625765
n_6 -6.941947 107.627582
n_7 -6.949322 107.625973
00000100
00100011
01010000
00101000
00010100

```


1 0 0 0 1 0 1 0
0 1 0 0 0 1 0 1
0 1 0 0 0 0 1 0



Gambar 3. Peta Buah-Batu

4. Bojonegoro.txt

```

22
n_0 -7.148304473331076 111.87785374025775
n_1 -7.149581924907767 111.8775747905127
n_2 -7.1483257642199565 111.87934504851017
n_3 -7.149730960692345 111.87925921781938
n_4 -7.148442864091083 111.88119040835487
n_5 -7.148581254806004 111.88273536075559
n_6 -7.148693031894448 111.88413010935197
n_7 -7.15228052962834 111.88329326014833
n_8 -7.150300489860512 111.88277827602346
n_9 -7.152237948218287 111.88255297046882
n_10 -7.152195366804266 111.8817804942815
n_11 -7.152046331823923 111.87921630249308
n_12 -7.1505453339528655 111.8793128620165
n_13 -7.150566624737297 111.88117967946916
n_14 -7.14929982133605 111.88117967946916
n_15 -7.149342403020049 111.88171612126591
n_16 -7.148501414026189 111.88191996914867
n_17 -7.15185471535077 111.87721001015157
n_18 -7.150449525412247 111.87748895988588
n_19 -7.150694369424705 111.88184486727543
n_20 -7.149289175916072 111.8824242244159
n_21 -7.150630497084131 111.88358293873883
0 1 1 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 1 0 0 0
1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

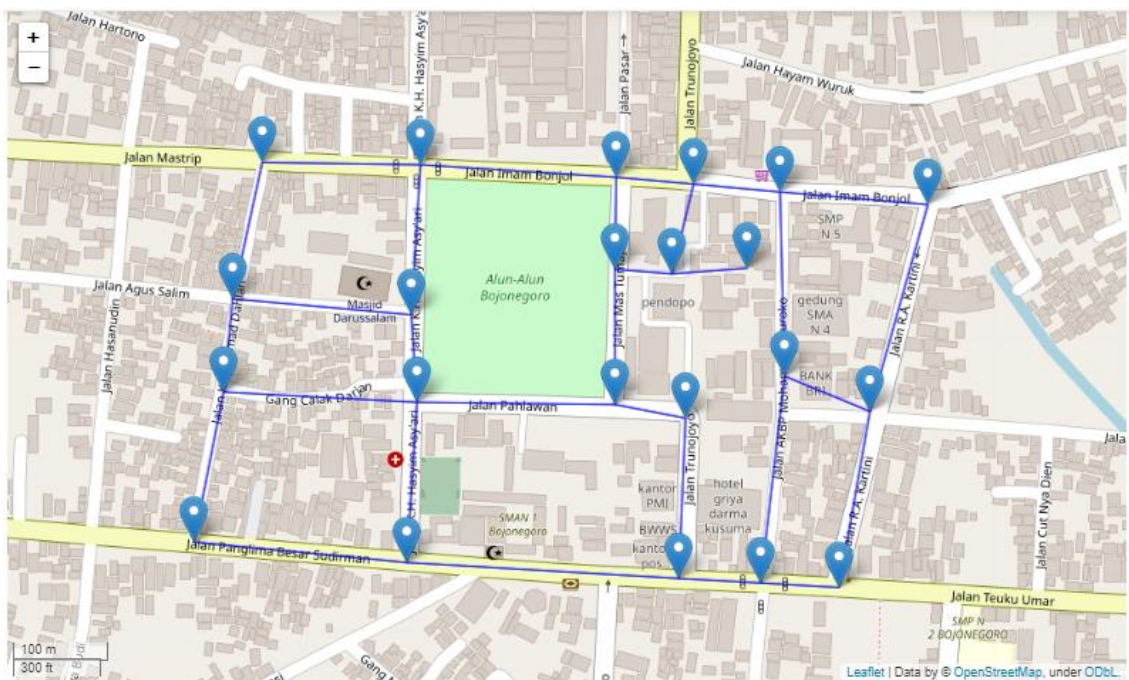
```



```

0110000000001000000000
00100000000000010100000
00000010100000000100000
00000100000000000000001
00000000010000000000001
00000100010000000000001
00000001101000000000000
00000000010100000000100
00000000001010000010000
000100000000101000001000
00000000000010100000100
00001000000000101000000
00000000000000101000010
00001100000000001000000
000000000000100000001000
0100000000000010000010000
0000000000001001000000000
0000000000000000001000000
000000111000000000000000

```



Gambar 4. Peta Bojonegoro

5. Bogor.txt

```

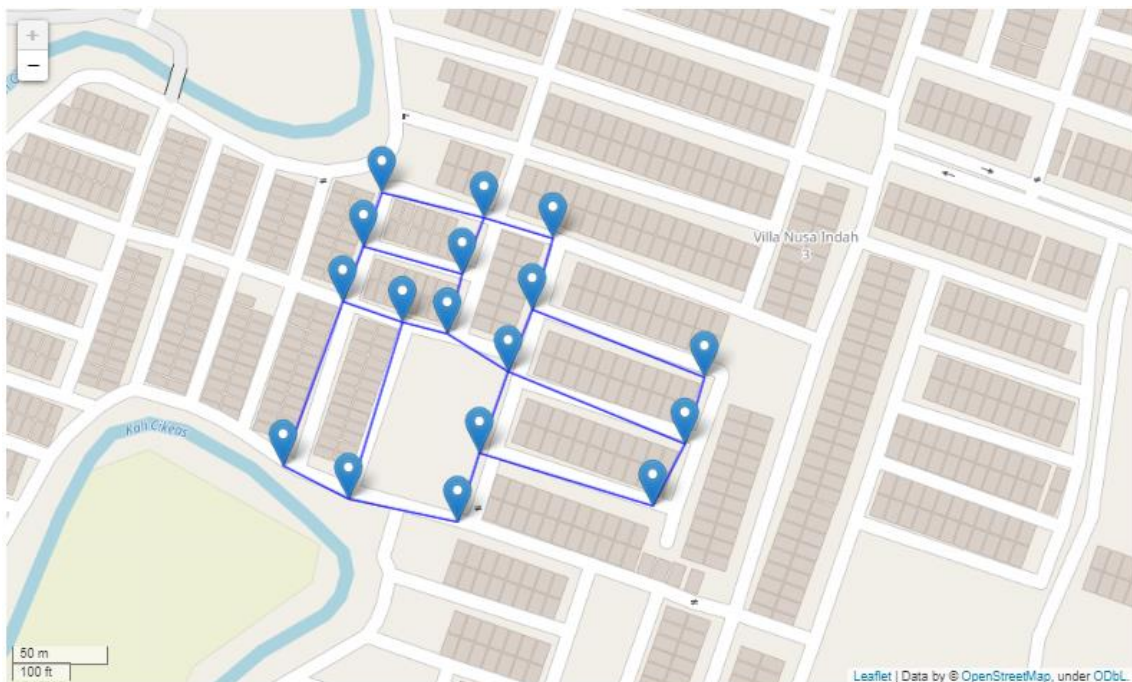
17
n_1 -6.331886362693373 106.962374434779
n_2 -6.331982767758166 106.96265351713453
n_3 -6.331629354810865 106.96247419809539
n_4 -6.331757315503621 106.96293553802147
n_5 -6.332034563562473 106.96286580059078
n_6 -6.332210509368757 106.96315011473128
n_7 -6.331922598018163 106.96326276750393

```

```

n_8 -6.331378765029025 106.96256002877931
n_9 -6.331496062388858 106.96303746195863
n_10 -6.331592032936168 106.96335932702334
n_11 -6.33223716781902 106.96406743016568
n_12 -6.332546405741545 106.96397623506402
n_13 -6.332834316744276 106.96382603136716
n_14 -6.332589059233552 106.96301600428765
n_15 -6.332908960311361 106.96291408035049
n_16 -6.332802326640789 106.9624044606647
n_17 -6.3326477077793255 106.96209868885325
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
1 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 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 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 1 1 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 1 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 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 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 1 0 1 0 0
0 0 0 0 0 1 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 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 0 0 0 0 0 1

```



Gambar 5. Peta Bogor

6. Jakarta.txt


```
18
n_1 -6.171043232785433 106.81100092687521
n_2 -6.167608534006892 106.82085261490053
n_3 -6.167901645940726 106.83181986111542
n_4 -6.171008622475124 106.82197292499775
n_5 -6.1722396836107105 106.82987405936763
n_6 -6.176343200067108 106.83117126053283
n_7 -6.171008622475124 106.83341188072724
n_8 -6.172943145832509 106.83423737237784
n_9 -6.182557035864119 106.83341188072724
n_10 -6.180563926670595 106.82273945295901
n_11 -6.183260484374661 106.8229753077163
n_12 -6.181794965588607 106.81684308402626
n_13 -6.184843240102121 106.81483831858914
n_14 -6.1821466904675795 106.81365904480258
n_15 -6.17546387778884 106.82055779645388
n_16 -6.175522499319491 106.82303427140563
n_17 -6.180345295173795 106.83218101371253
n_18 -6.179839688787598 106.81808132167109
0100000000000010000
1011000000000000000
0100001000000000000
01001000000000001100
0001011000000000000
0000100100000000010
0010100100000000000
0000011000000000000
0000000000100000010
000000000010000111
000000001101000000
000000000010110001
000000000001010000
1000000000001100000
0001000000000000101
000100000100001000
000001001100000000
000000000101001000
```




Gambar 6. Peta Jakarta

Berikut adalah hasil pengujian dengan beberapa scenario pengujian:

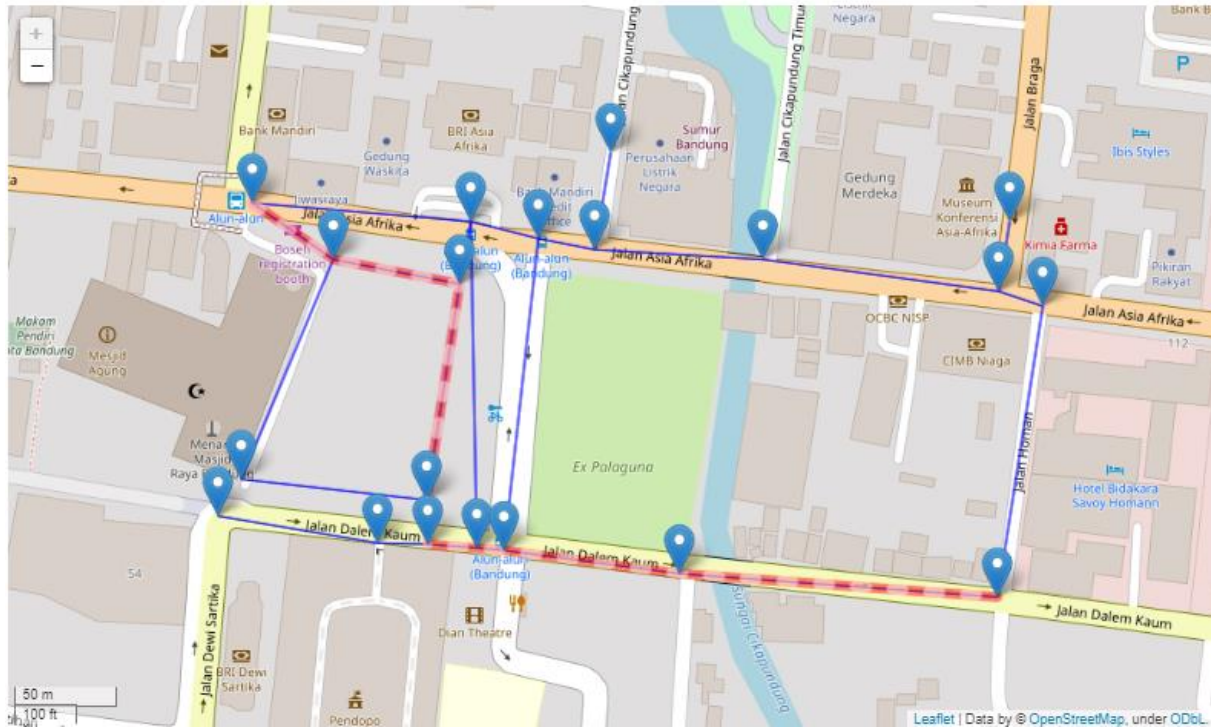
Testing 1	
File Graf	ITB.txt
Simpul Uji	n_2 dan n_18
Hasil Pengujian	
Panjang perjalanan yang harus ditempuh adalah 822 m	

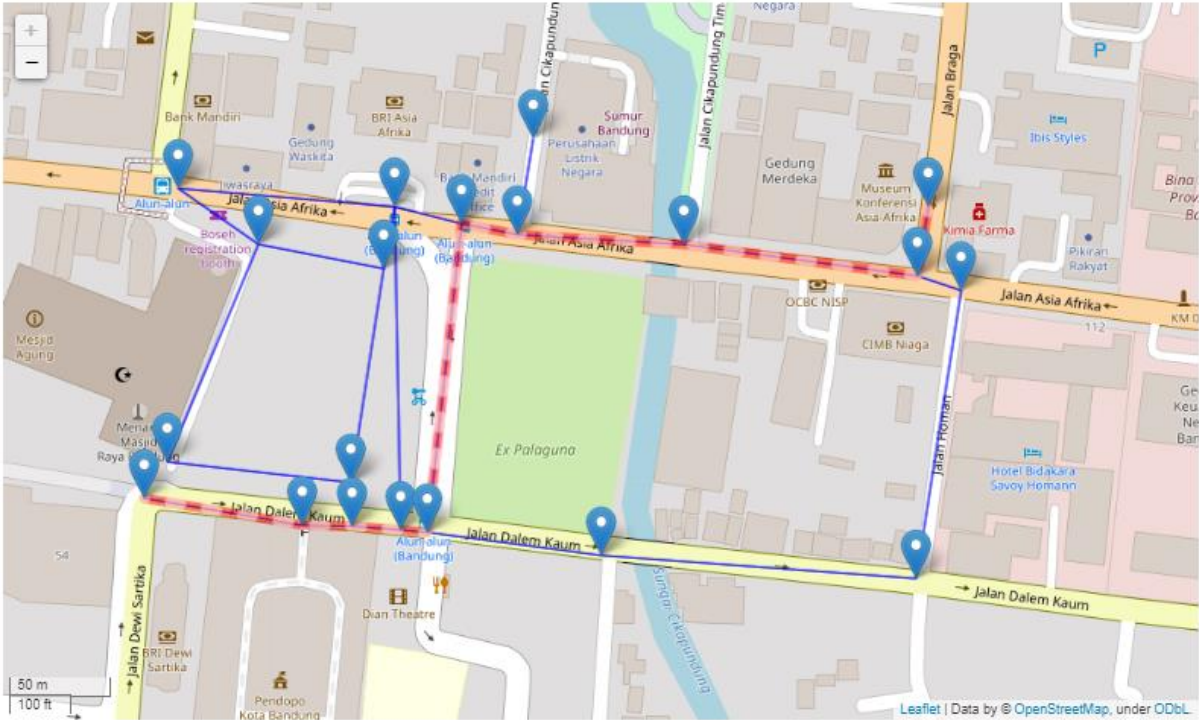
Testing 2	
File Graf	ITB.txt
Simpul Uji	n_5 dan n_15
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 659 m</p> 	

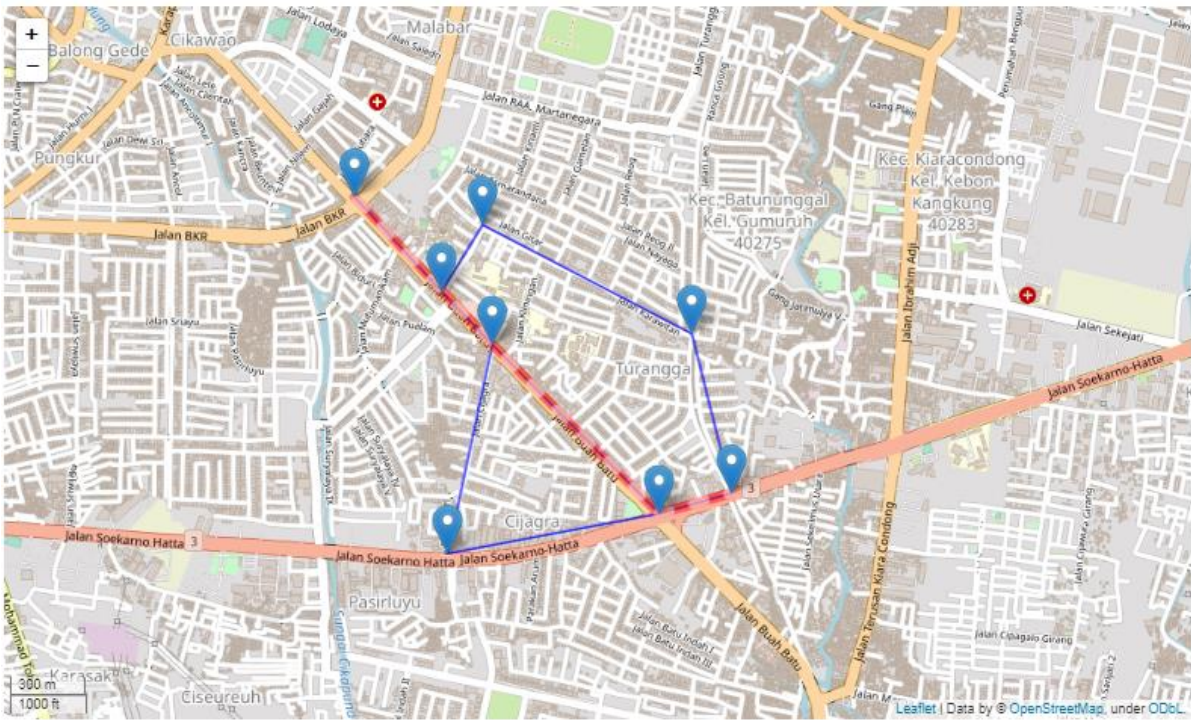
Testing 3	
File Graf	Alun-Alun.txt
Simpul Uji	n_0 dan n_14

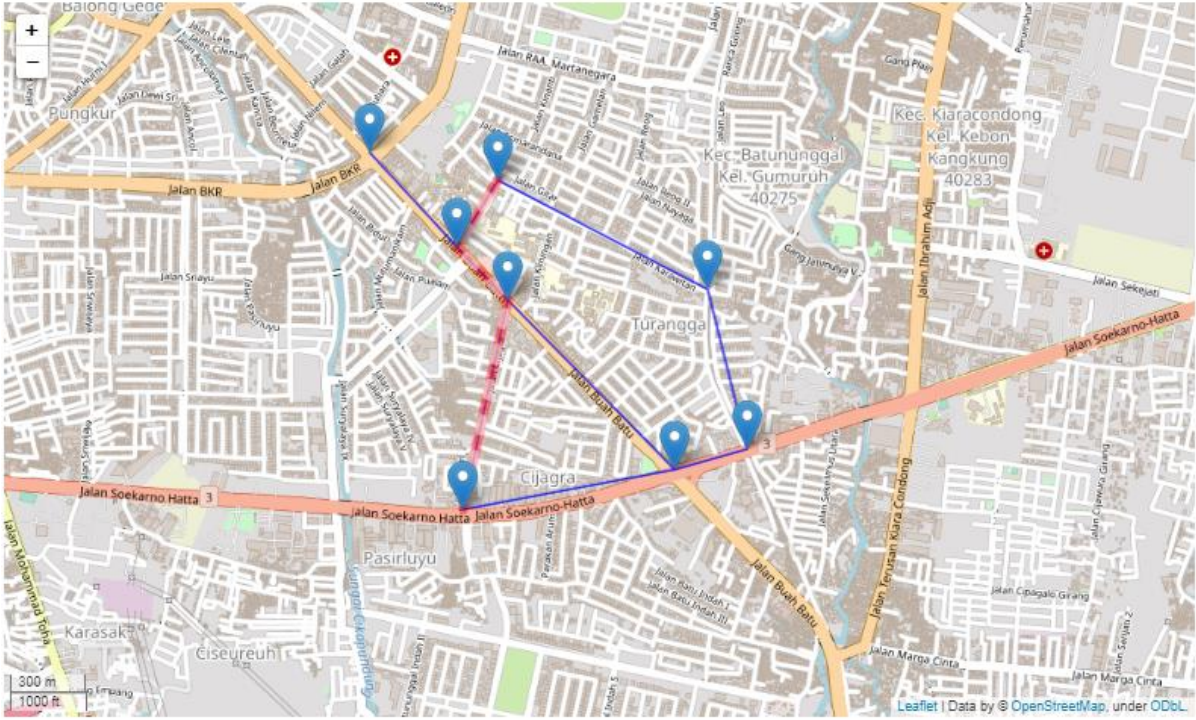
Hasil Pengujian

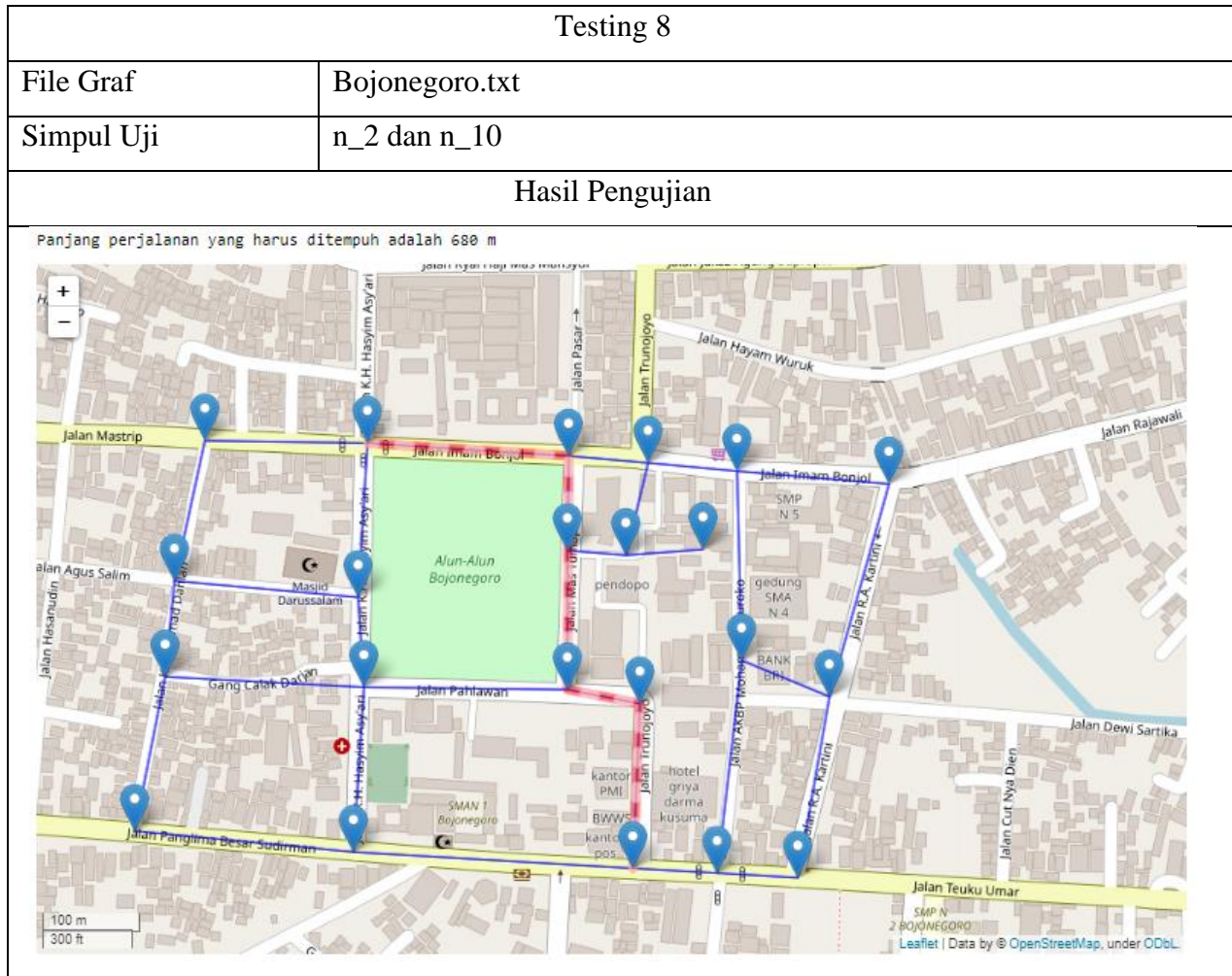
Panjang perjalanan yang harus ditempuh adalah 509 m

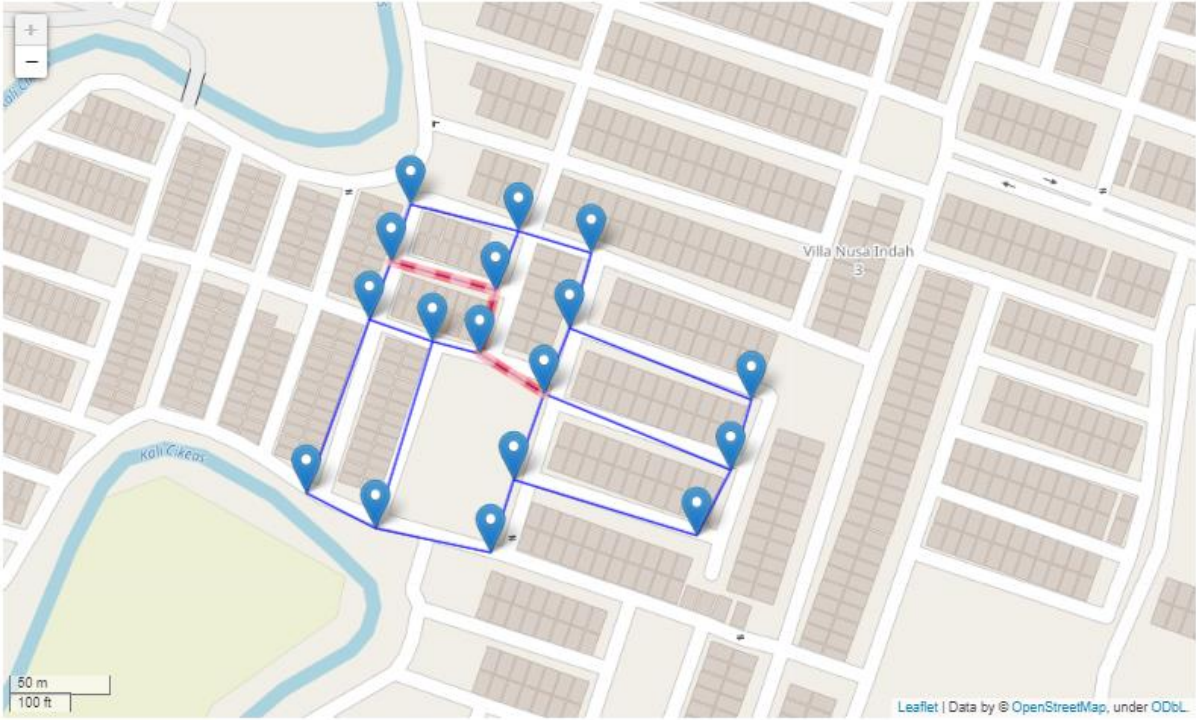


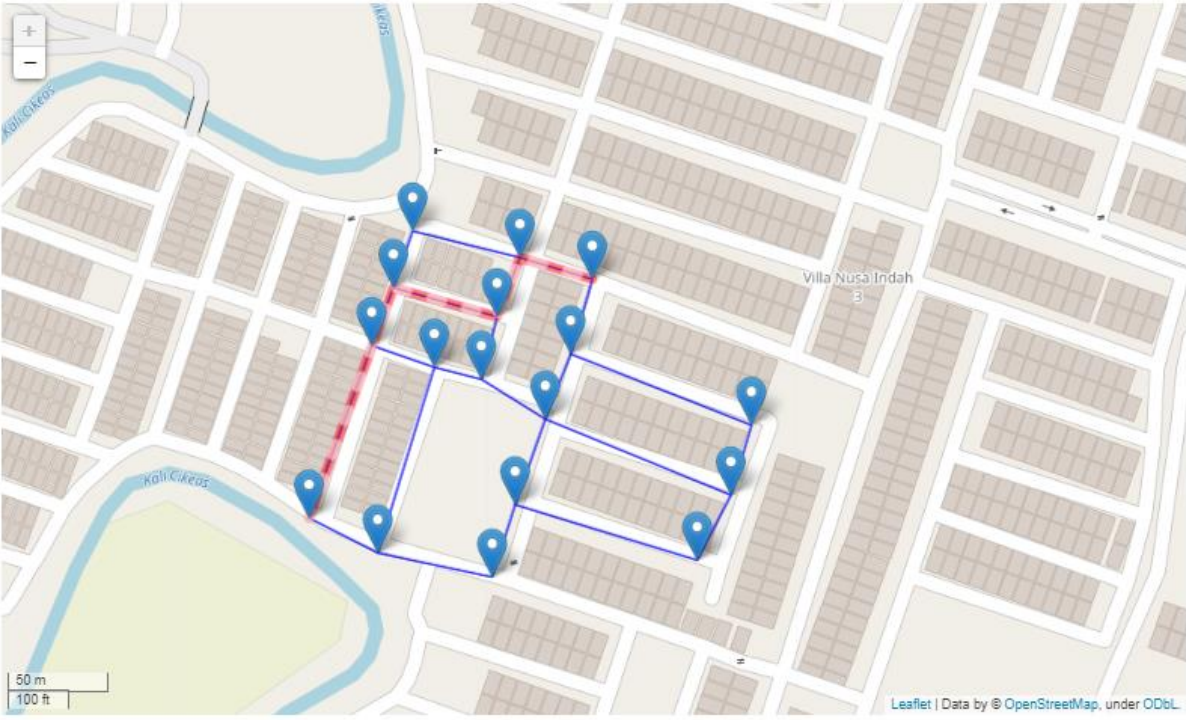
Testing 4	
File Graf	Alun-Alun.txt
Simpul Uji	n_16 dan n_17
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 545 m</p> 	


Testing 5	
File Graf	Buah-Batu.txt
Simpul Uji	n_0 dan n_2
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 2010 m</p> 	


Testing 6	
File Graf	Buah-Batu.txt
Simpul Uji	n_4 dan n_7
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 1432 m</p> 	



Testing 9	
File Graf	Bogor.txt
Simpul Uji	n_3 dan n_6
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 120 m</p> 	

Testing 10	
File Graf	Bogor.txt
Simpul Uji	n_10 dan n_17
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 239 m</p> 	

Testing 11	
File Graf	Jakarta.txt
Simpul Uji	n_1 dan n_6
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 2913 m</p> 	

Testing 12	
File Graf	Jakarta.txt
Simpul Uji	n_3 dan n_13
Hasil Pengujian	
<p>Panjang perjalanan yang harus ditempuh adalah 3346 m</p> 	

LAMPIRAN

1. Alamat Drive Kode Program

<https://github.com/AndhikaRei/tucil3stima>

2. Cek list Tugas

Poin	Ya	Tidak
1. Program dapat menerima input graf	✓	
2. Program dapat menghitung lintasan terpendek	✓	
3. Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
4. Bonus: Program dapat menerima input peta dengan Google Map Api dan menampilkan peta		✓ (hanya bisa visualisasi peta dan rute dari 2 simpul pilihan)