Laporan Tugas Kecil Strategi Algoritma
Implementasi Algoritma A* untuk Mengetahui Lintasan Terpendek
Semester II 2020/2021

Oleh :
Faris Hasim Syauqi (13519050)
Randy Zakya Suchrady (13519061)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

DAFTAR ISI

KODE PROGRAM

Berikut kode program dari algoritma A* untuk mendapatkan lintasan terpendek yang ditulis dengan bahasa Python. (Dapat dilihat di https://github.com/rdyzakya/Tucil3Stima)

modul graph

node_exception.py

```
class NullNodeException(Exception):
	pass
```

---

node.py

```
from graph import node_exception as ne

class Node:
	#ctor
	def __init__(this,value):
		this.name = value
		this.neighbor = []

	#exception null node
	def check_null_node(this):
		if (this.name == "NULL"):
			raise ne.NullNodeException("Violate a Null Node Value")
			##null node itu Node("NULL")

	#mendapatkan nama node
	def get_name(this):
		this.check_null_node()
		return this.name

	#mengganti nama node
	def set_name(this,new_name):
		this.check_null_node()
		this.name = new_name
		return

	#mendapatkan neighbor pada indeks tertentu
	def get_neighbor(this,neighbor_name):
		this.check_null_node()
		for it in this.neighbor:
```

```
            if (it.name == neighbor_name):
                  return it
      return Node("NULL")

#mendapatkan seluruh list neighbor
def get_all_neighbor(this):
      return this.neighbor

#mendapatkan panjang list neighbor
def get_neighbor_num(this):
      this.check_null_node()
      return len(this.neighbor)

#mengganti neighbor pada indeks tertentu di list neighbor
def set_neighbor(this,i,value):
      this.check_null_node()
      if (i < len(this.neighbor)):
            neighbor_node = Node(value)
            this.neighbor[i] = neighbor_node
            return True
      return False

#menambah neighbor
def add_neighbor(this,neighbor_node):
      this.check_null_node()
      this.neighbor.append(neighbor_node)
      return

#menghapus neighbor dengan nama tertentu
def del_neighbor(this,neighbor_name):
      this.check_null_node()
      i = 0
      neighbor_node = Node("NULL")
      found = False
      while(i < len(this.neighbor) and not(found)):
            if (this.neighbor[i].name != neighbor_name):
                  i += 1
            else:
                  neighbor_node = this.neighbor[i]
                  this.neighbor.remove(neighbor_node)
                  found = True
      return neighbor_node
```

```
	#mengecek keberadaan neighbor
	def is_exist_neighbor(this,neighbor_name):
		for it in this.neighbor:
			if (it.name == neighbor_name):
				return True
		return False

	#mengecek null node
	def is_null(this):
		return this.name == "NULL"

	#untuk debugging
	def print_all(this):
		this.check_null_node()
		print("Node name: " + this.name)
		print("Neighbor(s) : ")
		for i in range(len(this.neighbor)):
			print("- " + this.neighbor[i].name)
		return
```

---

graph.py

```
from graph import node

class Graph:
	#ctor
	def __init__(this):
		this.nodes = []

	#menambah node
	def add_node(this,node): ##typenya Node atau subclassnya aja
		this.nodes.append(node)
		return

	#mendapatkan indeks tertentu sebuah node dengan param nama
	def get_node_idx(this,node_name):
		for i in range(len(this.nodes)):
			if (this.nodes[i].get_name() == node_name):
				return i
		return -1
```

```
	#mendapatkan return value dengan tipe kelas Node menggunakan param nama
	def get_node(this,node_name):
		idx = this.get_node_idx(node_name)
		if (idx != -1):
			return this.nodes[idx]
		return node.Node("NULL")

	#mengcek apakah dua node dengan nama tertentu bersisian
	def is_exist_edge(this,node1_name,node2_name):
		first = this.get_node(node1_name).is_exist_neighbor(node2_name)
		second = this.get_node(node2_name).is_exist_neighbor(node1_name)
		return first or second

	#menambah sisi
	def add_edge(this,node1_name,node2_name):
		node1_idx = this.get_node_idx(node1_name)
		node2_idx = this.get_node_idx(node2_name)
		if (node1_idx != -1 and node2_idx != -1):
			if (not(this.is_exist_edge(node1_name,node2_name))):

this.nodes[node1_idx].add_neighbor(this.nodes[node2_idx])

this.nodes[node2_idx].add_neighbor(this.nodes[node1_idx])
				return True
		return False

	#menambah sisi banyak sekaligus
	def add_many_edge(this,node_name,list_node_name):
		for i in list_node_name:
			this.add_edge(node_name,i)

	#menghilangkan sebuah sisi
	def del_edge(this,node1_name,node2_name):
		node1_idx = this.get_node_idx(node1_name)
		node2_idx = this.get_node_idx(node2_name)
		if (node1_idx != -1 and node2_idx != -1):
			if(this.is_exist_edge(node1_name,node2_name)):
				this.nodes[node1_idx].del_neighbor(node2_name)
				this.nodes[node2_idx].del_neighbor(node1_name)
				return True
		return False
```

```
#menghilangkan sebuah node
def del_node(this,node_name):
	for i in range(len(this.nodes)):
		this.nodes[i].del_neighbor(node_name)
	idx = this.get_node_idx(node_name)
	if (idx != -1):
		value = this.nodes[idx]
		this.nodes.remove(this.nodes[idx])
		return value
	return node.Node("NULL")

#untuk mencari keterhubungan dua buah node
def dfs_search(this,node1_name,node2_name):
	node1_idx = this.get_node_idx(node1_name)
	if (node1_idx == -1 or this.get_node_idx(node2_name) == -1):
		return False
	visited = [0 for i in range(len(this.nodes))]
	visited[node1_idx] = 1
	record = [node1_name]
	while(node1_name != node2_name):
		node1 = this.get_node(node1_name)
		list_neighbor = node1.get_all_neighbor()
		found = False
		i = 0
		while(i < len(list_neighbor) and not found):
			neighbor_name = list_neighbor[i].get_name()
			neighbor_idx = this.get_node_idx(neighbor_name)
			if (visited[neighbor_idx] == 0):
				visited[neighbor_idx] = 1
				record.append(neighbor_name)
				node1_name = neighbor_name
				found = True
			else:
				i += 1
		if (node1_name == node2_name):
			return True
		if (i == len(list_neighbor)):
			length = len(record)
			if (length == 1):
				return False
			else:
				record.pop(length-1)
```

```
                        node1_name = record[length-2]

      #untuk debugging
      def print_all(this):
            for it in this.nodes:
                  it.print_all()
```

---

coordinate2d.py

```
import math

class Coordinate2D:
      #ctor
      def __init__(this,x,y):
            this.x = x
            this.y = y

      #mendapatkan jarak dengan formula haversine
      def haversine_distance(this,another_point):
            lat1 = math.radians(this.x)
            lat2 = math.radians(another_point.x)
            lon1 = math.radians(this.y)
            lon2 = math.radians(another_point.y)

            #Haversine
            delta_lat = lat2 - lat1
            delta_lon = lon2 - lon1
            a = math.sin(delta_lat/2)**2 + math.cos(lat1) * math.cos(lat2) *
math.sin(delta_lon/2)**2
            c = 2 * math.asin(math.sqrt(a))
            earth_r = 6371
            return (c * earth_r * 1000)
```

---

location.py

```
from graph import node
import coordinate2d as c2d

#derived class of Node
class Location(node.Node):
      #ctor
      def __init__(this,name,x,y):
```

```
            super().__init__(name)
            this.coordinate = c2d.Coordinate2D(x,y)
      #memperoleh jarak antara dua buah lokasi
      def distance(this,another_location):
            return
this.coordinate.haversine_distance(another_location.coordinate)
```

---

prioqueue.py

```
class ElementQueueMap:
      #ctor
      def __init__(this,track,target):
            this.track = track
            this.prio = 0

            length = len(this.track)
            if (length > 0):
                  i = 0
                  while(i+1 < length):
                        temp_dist = this.track[i].distance(this.track[i+1])
                        this.prio += temp_dist
                        i += 1
                  this.prio += this.track[length-1].distance(target)

class PrioQueueMap:
      #ctor
      def __init__(this):
            this.queue = []
            this.flag = True

      #sort queue
      def sort_queue(this):
            length = len(this.queue)
            for i in range(length):
                  for j in range(i+1,length):
                        if (this.queue[j].prio < this.queue[i].prio):
                              temp = this.queue[i]
                              this.queue[i] = this.queue[j]
                              this.queue[j] = temp
      #nambah element queue
      def enqueue(this,element):
            this.queue.append(element)
```

```
            this.sort_queue()
      #mengeluarkan element queue
      def dequeue(this):
            if (len(this.queue) > 0):
                  result = this.queue.pop(0)
                  return result
            this.flag = False
            return None
      #mendapatkan panjang queue
      def get_size(this):
            return len(this.queue)
```

---

map.py

```
import location as loc
import prioqueue as pq
from graph import graph
from graph import node

#derived class of Graph
class Map(graph.Graph):

      #ctor
      def __init__(this):
            super().__init__()

      #algoritma a star alternatif 1 dengan dfs untuk cari keterhubungan dulu
      def a_star_path(this,origin,target):
            if (origin.is_null() or target.is_null()):
                  return ([],-1)
            if (this.dfs_search(origin.get_name(),target.get_name())):
                  queue = pq.PrioQueueMap()
                  initial_element = pq.ElementQueueMap([origin],target)
                  queue.enqueue(initial_element)
                  firstdeq = queue.dequeue()
                  cur = firstdeq.track #bentuknya list node
                  distance = firstdeq.prio
                  while(cur[len(cur)-1].get_name()  !=  target.get_name()  and
queue.flag):
                        neighbor = cur[len(cur)-1].get_all_neighbor()
                        new_track = cur[:]
                        i = 0
```

```
                        for it in neighbor:
                              new_track.append(it)
                              new_elqueue                                     =
pq.ElementQueueMap(new_track,target)
                              queue.enqueue(new_elqueue)
                              new_track = cur[:]
                        new_dequeue = queue.dequeue()
                        cur = new_dequeue.track
                        distance = new_dequeue.prio
                  if (not(queue.flag)):
                        return ([],-1)
                  return (cur,distance)
            return ([],-1)

      # alternatif 2 tanpa dfs
      def a_star_path2(this,origin,target):
            if (origin.is_null() or target.is_null()):
                  return ([],-1)
            visited_dict = {}
            for nodes in this.nodes:
                  visited_dict[nodes.get_name()] = False
            queue = pq.PrioQueueMap()
            initial_element = pq.ElementQueueMap([origin],target)
            queue.enqueue(initial_element)
            cur = initial_element.track
            distance = initial_element.prio
            while(cur[len(cur)-1].get_name()    !=    target.get_name()    and
len(queue.queue) != 0):
                  new_dequeue = queue.dequeue()
                  cur = new_dequeue.track #bentuknya list node
                  distance = new_dequeue.prio
                  visited_dict.update({cur[len(cur)-1].get_name()  :  True})  #
tandai sudah dikunjungi
                  neighbor = cur[len(cur)-1].get_all_neighbor()
                  new_track = cur[:]
                  for it in neighbor:
                        if not visited_dict.get(it.get_name()):
                              new_track.append(it)
                              new_elqueue                                     =
pq.ElementQueueMap(new_track,target)
                              queue.enqueue(new_elqueue)
                              new_track = cur[:]
```

```
            if (len(queue.queue) == 0):
                  return ([],-1)
            return (cur,distance)
```

---

mapLoader.py

```
import map
import location as loc

class MapLoader:
    def __init__(self):
        # test
        self.test = True

    def load(self, filename):
        mapLoaded = map.Map()
        f = open(filename, 'r')

        countNodes = int(f.readline())
        nodes = ["X" for i in range(countNodes)]

        # read location/node
        for i in range(countNodes):
            lines = f.readline().split(' ')
            nodes[i] = lines[0]
                    mapLoaded.add_node(loc.Location(lines[0],  float(lines[1]),
float(lines[2])))

        # read edge
        for i in range(countNodes):
            lines = f.readline().split(' ')
            this_name = nodes[i]
            for j in range(countNodes):
                other_name = nodes[j]
                if lines[j] == "1":
                    mapLoaded.add_edge(this_name, other_name)

        return mapLoaded
```

---

mapVisualizer.py

```
import map
```

```
import folium

class MapVisualizer:
    def visualize(self, mapSolved, path):
        self.visualMap = folium.Map(location=[mapSolved.nodes[0].coordinate.x,
                                    mapSolved.nodes[0].coordinate.y],
                                zoom_start=17)

        # create marker for each node
        self.visualizeMap(mapSolved)

        # create marker for each node in path
        self.visualizePath(path)

        print("You can see the result in : " + "testscmap.html")
        self.visualMap.save('testscmap.html')

    def visualizeMap(self, mapSolved):
        for node in mapSolved.nodes:
            folium.Marker([node.coordinate.x,
                           node.coordinate.y],
                           popup='<strong>'+node.get_name()+'</strong>',
                                        icon=folium.Icon(icon='leaf',
color='blue')).add_to(self.visualMap)

        for node1 in mapSolved.nodes:
            for node2 in mapSolved.nodes:
                               if  mapSolved.is_exist_edge(node1.get_name(),
node2.get_name()):
                    folium.PolyLine([[node1.coordinate.x, node1.coordinate.y],
                                     [node2.coordinate.x, node2.coordinate.y]],
                                     color='blue').add_to(self.visualMap)

    def visualizePath(self, path):
        countNodes = len(path)
        for i in range(countNodes):
            if i != countNodes-1:
                node1 = path[i]
                node2 = path[i+1]
                folium.PolyLine([[node1.coordinate.x, node1.coordinate.y],
                                 [node2.coordinate.x, node2.coordinate.y]],
                                 color='red').add_to(self.visualMap)
```

main.py

```
import folium
import map
import location as loc
import mapLoader as ml
import mapVisualizer as mv
import os
import webbrowser

filename = input("Masukkan nama file txt : ")

peta = ml.MapLoader().load("../test/" + filename)

print("Simpul-simpul yang tersedia : ")
for node in peta.nodes:
    print(node.get_name(), end=" ")
print()

node1 = input("Masukkan simpul awal : ")
node2 = input("Masukkan simpul tujuan : ")

result = peta.a_star_path2(peta.get_node(node1),peta.get_node(node2))

if (len(result[0]) != 0):
    for i in range(len(result[0])):
        print(result[0][i].get_name() , end = "")
        if (i != len(result[0]) -1):
            print(" --> " , end = "")
        else:
            print()
    # for i in result[0]:
    #     print(i.get_name() , end = "")
    print("%.2f" % result[1] + " metres")
    mv.MapVisualizer().visualize(peta, result[0], result[1])
    url = 'file:///' + os.getcwd() + '/' + 'testscmap.html'
    webbrowser.open(url)

else:
    print("not connected")
```

INPUT GRAF/PETA

Berikut adalah beberapa input yang kami gunakan dan bisa digunakan sebagai contoh input program ini. Input ditulis dalam format txt.

1. Peta jalan sekitar kampus ITB/Dago

```
13
A -6.893837 107.608466
B -6.893251 107.61044
C -6.893794 107.61294
D -6.893656 107.611942
E -6.894742 107.611706
F -6.894806 107.610204
G -6.894902 107.608842
H -6.887446 107.613595
I -6.892015 107.608016
J -6.887787 107.60823
K -6.887734 107.610312
L -6.887297 107.611395
M -6.893294 107.609872
0 0 0 0 0 0 1 0 1 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 1 0 0 0 0 0
0 1 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 0 0 1 0 1 0 0 0 0 0 1
1 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
1 0 0 0 0 0 0 0 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 0 0 1 0 1 0
0 0 0 0 0 0 0 1 0 0 1 0 0
1 1 0 0 0 1 0 0 0 0 0 0 0
```

2. Peta jalan sekitar Alun-alun Bandung

```
14
dalem_kaum -6.922053 467.604049
a -6.920754 467.604145
```

```
b -6.918315 467.604295
angkringan_losari -6.919007 467.606676
bank_mandiri -6.92096 467.60643
c -6.921261 467.607742
d -6.922475 467.607674
e -6.92242 467.606395
f -6.923379 467.606266
g -6.923102 467.60396
h -6.923012 467.603264
i -6.9229 467.602608
j -6.921934 467.602721
k -6.921996 467.603357
0 1 0 0 0 0 0 0 0 1 0 0 0 1
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 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 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 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 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 1
1 0 0 0 0 0 0 0 0 0 1 0 1 0
```

3. Peta jalan sekitar Buahbatu

```
17
A -6.987429 467.644773
B -6.993989 467.642283
C -6.981508 467.636962
D -6.98892 467.652454
E -6.978015 467.653699
F -6.977163 467.649965
G -6.991689 467.668719
```

```
H -6.990155 467.668462
I -6.997354 467.668891
J -6.99893 467.668247
K -6.998632 467.664127
L -6.999356 467.663999
M -6.998589 467.660222
N -7.003573 467.657433
O -6.986098 467.644011
P -6.998866 467.641811
Q -7.007257 467.641296
0 1 0 1 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 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
1 0 0 0 1 0 1 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 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 1 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 1 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 1 0 1 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 1 0 0 1 0
1 0 1 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 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

4. Peta jalan sebuah kawasan di kotamu (Perumahan Kota Wisata, Kabupaten Bogor)

```
10
gerbang_kotwis -6.349855 466.960423
bunderan_kowis -6.351781 466.963664
a -6.352627 466.963115
b -6.353429 466.962153
c -6.355252 466.961411
d -6.356402 466.962008
```

```
e -6.358854 466.961297
f -6.359679 466.963712
g -6.357515 466.964472
h -6.353704 466.965138
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 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 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 1 0 1
0 1 0 0 0 0 0 0 1 0
```

5. Peta area Monumen Nasional

```
21
A -6.18186 106.81367
B -6.18175 106.81664
C -6.18493 106.81487
D -6.18345 106.82294
E -6.1826 106.83329
F -6.17873 106.84188
G -6.16388 106.83833
H -6.15928 106.83724
I -6.16252 106.82877
J -6.16002 106.81899
K -6.16755 106.82075
L -6.16791 106.83105
M -6.17104 106.82189
N -6.1764 106.83108
O -6.17228 106.83408
P -6.17282 106.83008
Q -6.17096 106.8334
R -6.19925 106.85389
S -6.18684 106.83612
```

```
T -6.18691 106.82306
U -6.19491 106.82302
0 1 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 1 0 0 0 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
0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 1 0 1 0 0 0 0 0 0 0 1 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 1 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
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 1 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 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 1 0 1 0 0 0 0 1 1 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 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 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 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 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 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 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

6. Peta di daerah kota Blitar

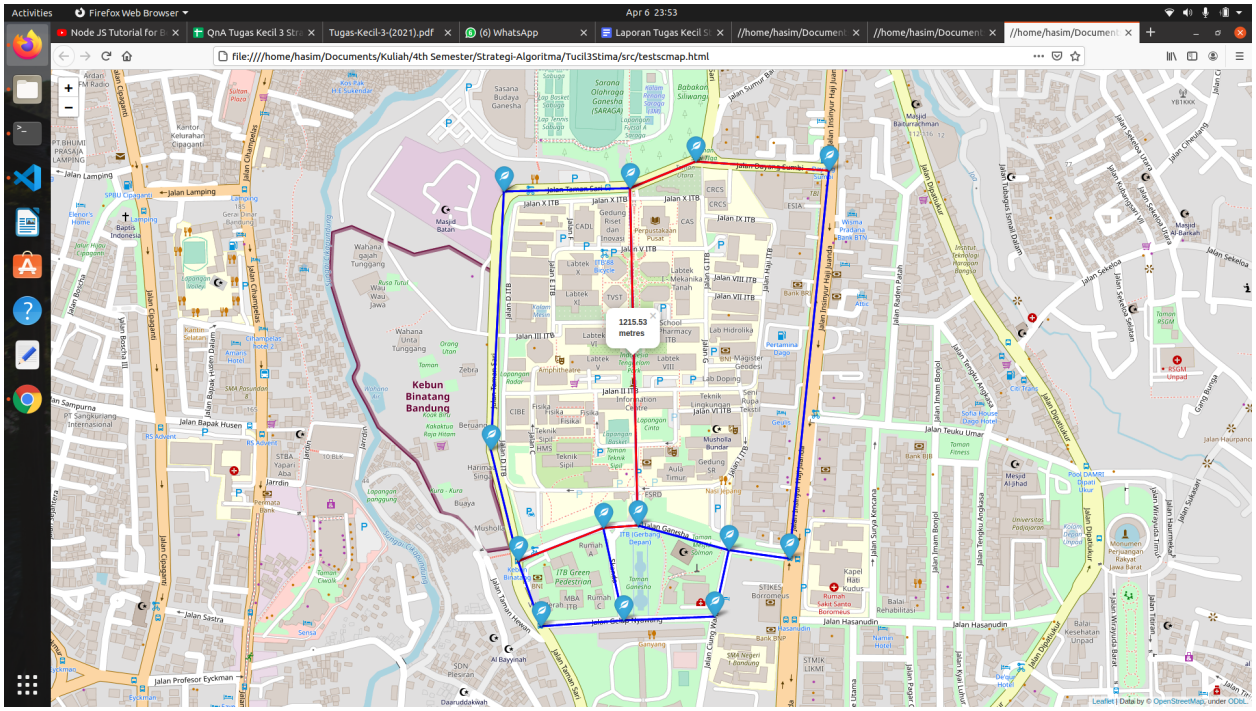
```
11
A -8.099045120315456 112.16493522999168
B -8.100840204992812 112.16458117838853
C -8.100340981063543 112.16244613993318
D -8.098853927392977 112.16268217433529
E -8.099013254834688 112.16429149980415
F -8.099066363967909 112.16586863876363
G -8.099140716742642 112.16737067586789
H -8.097600549319546 112.16749942190542
I -8.097419977580683 112.16605102898342
J -8.097165052635042 112.16450607653333
```

```
K -8.096888883761814 112.1628645645551
0 1 0 0 1 1 0 0 0 0 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 1 0 1 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0 1 0
0 0 0 0 0 0 0 0 1 0 1
0 0 0 1 0 0 0 0 0 1 0
```

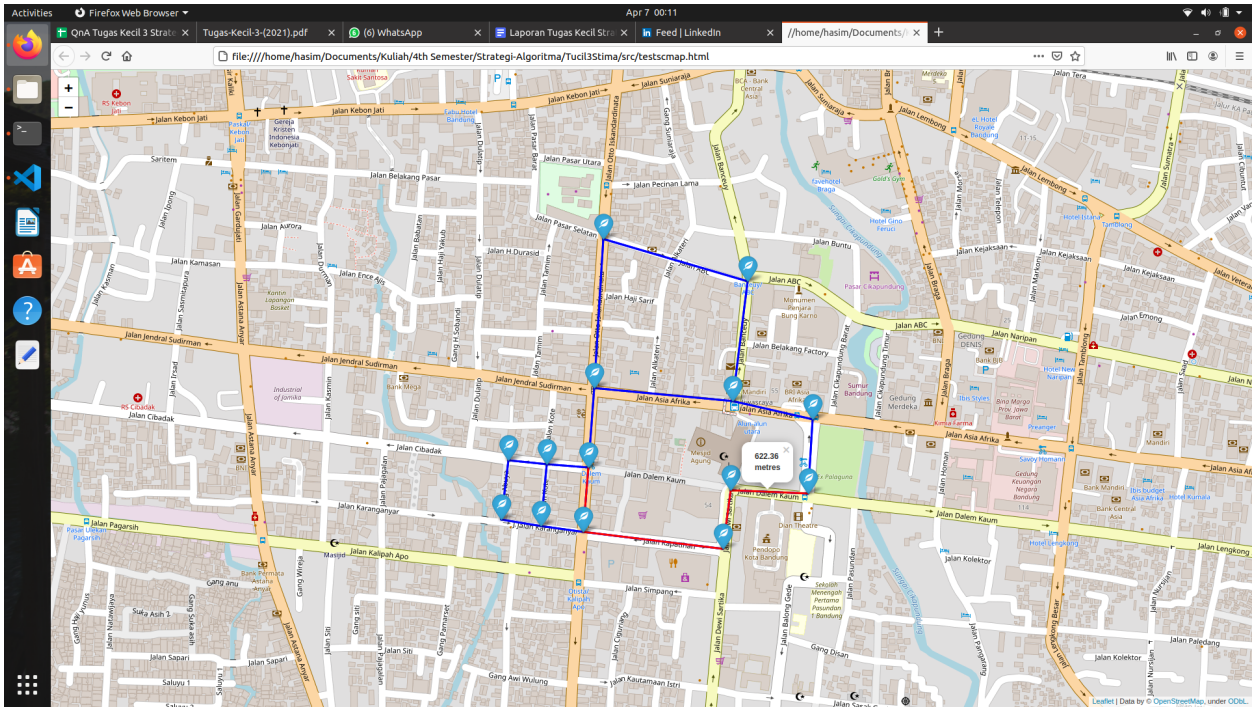
TEST CASE

Berikut adalah hasil screenshot dari beberapa test case.

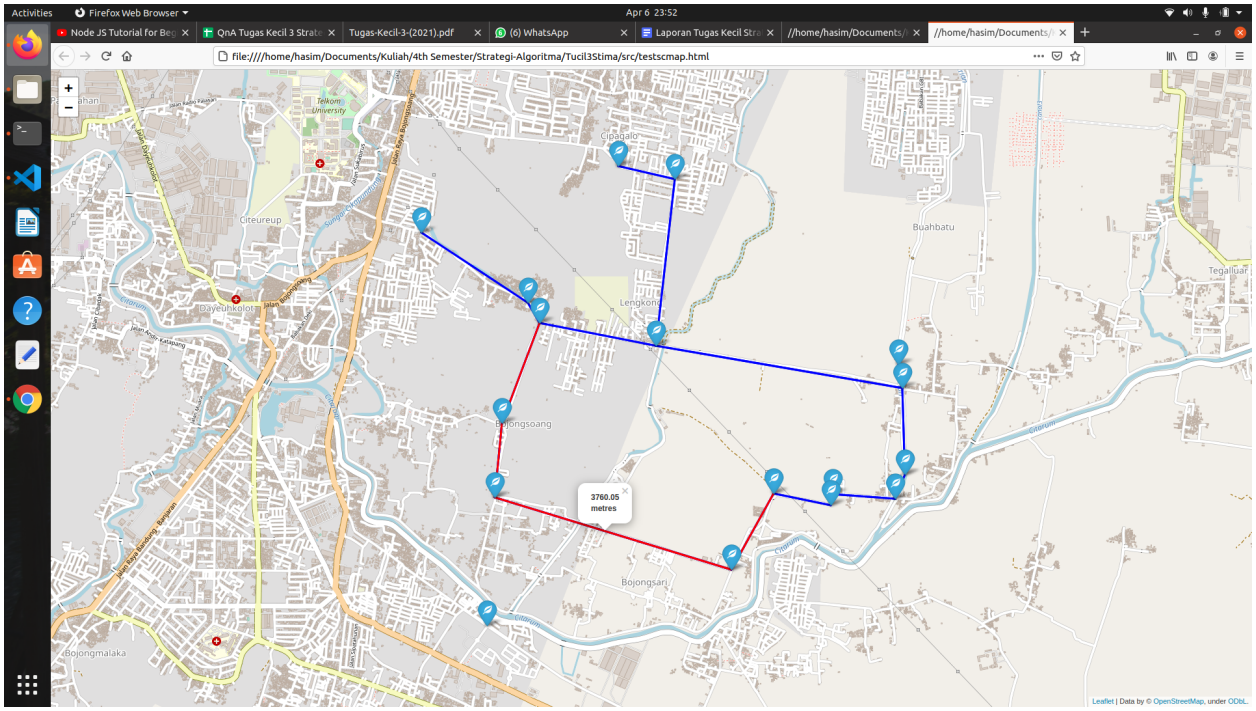
1. Peta jalan sekitar kampus ITB/Dago



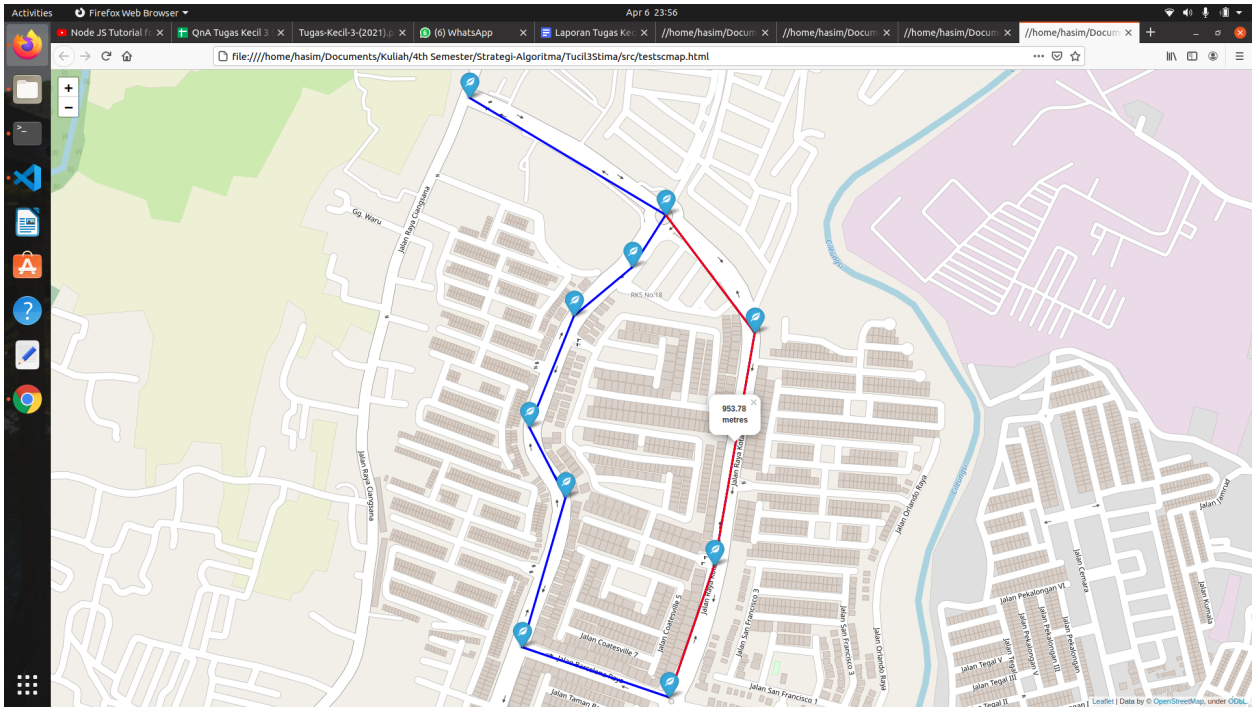
2. Peta jalan sekitar Alun-alun Bandung



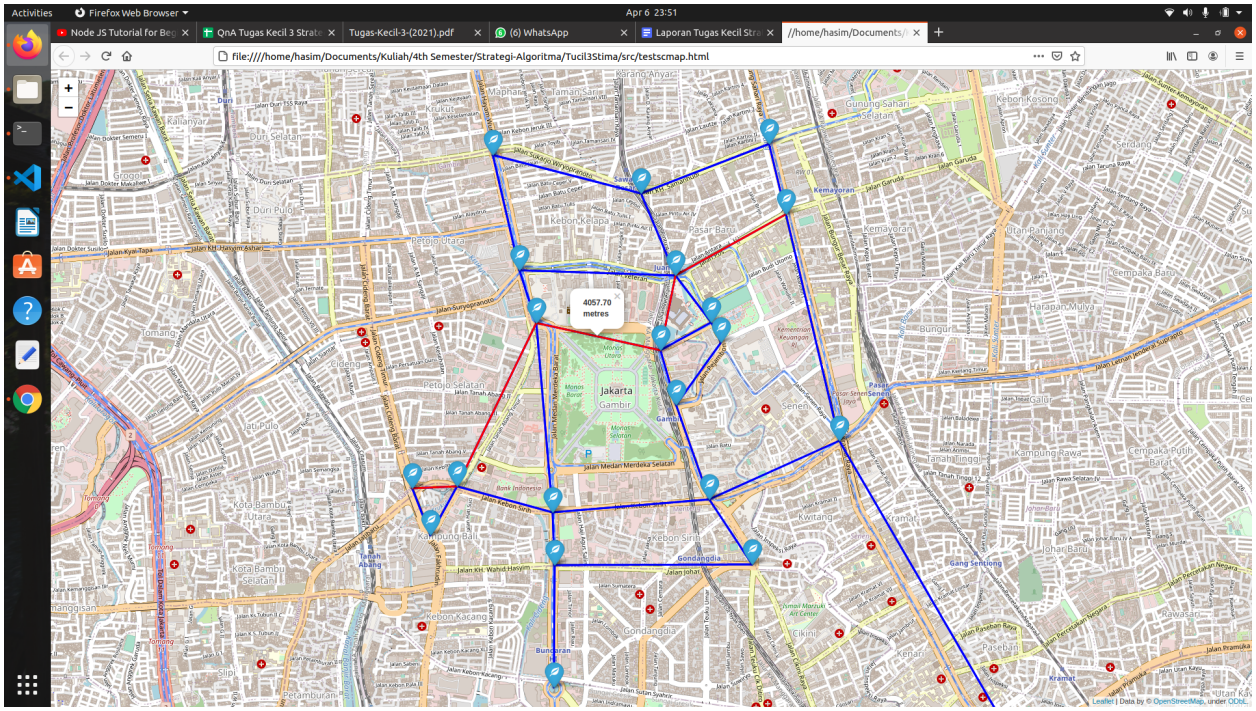
3. Peta jalan sekitar Buahbatu



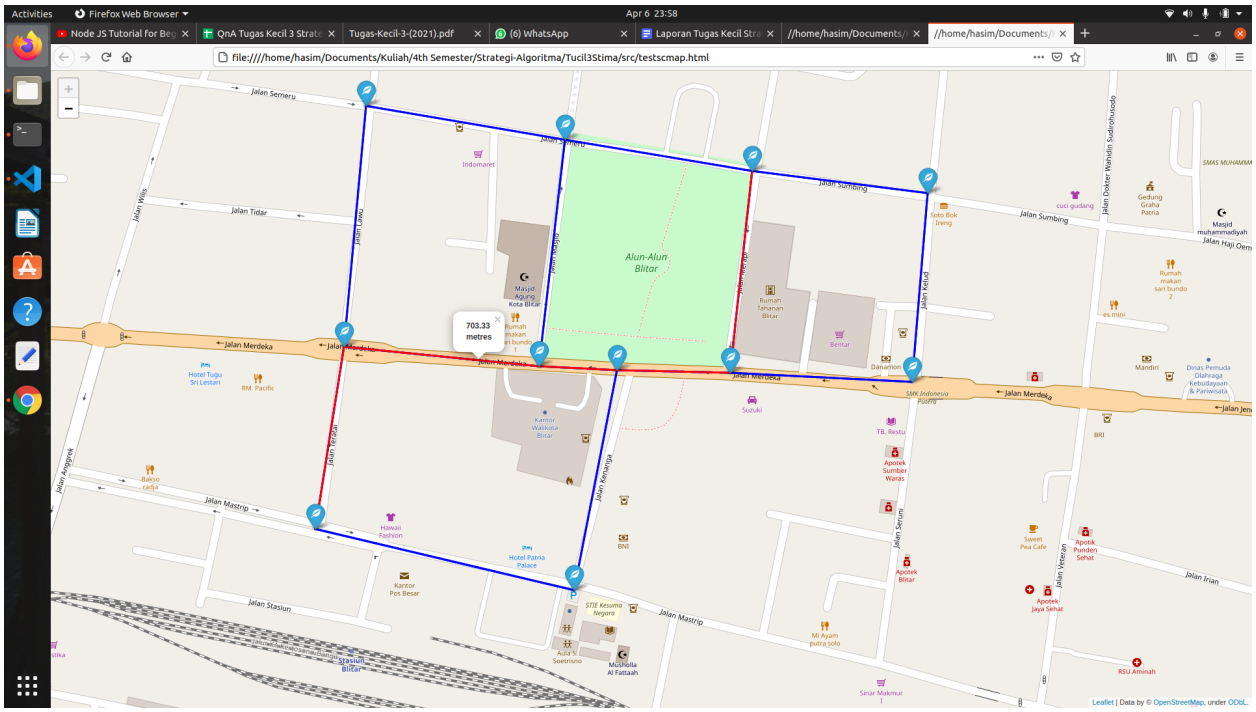
4. Peta jalan sebuah kawasan di kotamu (Perumahan Kota Wisata, Kabupaten Bogor)



5. Peta area Monumen Nasional



6. Peta di daerah kota Blitar



TABEL PENILAIAN

1	Program dapat menerima input graf	✔
2	Program dapat menghitung lintasan terpendek	✔
3	Program dapat menampilkan lintasan terpendek beserta jaraknya	✔
4	Bonus : Program dapat menerima input peta dengan Google Map API (atau semisalnya) dan menampilkan peta	✔