

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Disusun Oleh :

*Daru Bagus Dananjaya - 13519080
Shifa Salsabiila - 13519106*

Deskripsi Tugas

Pada tugas kali ini, mahasiswa diminta membuat sebuah aplikasi sederhana yang dapat menentukan lintasan terpendek dari suatu titik ke titik lain berdasarkan peta Google Map. Ruas-ruas jalan di peta dibentuk menjadi sebuah graf. Simpul pada graf menyatakan persimpangan atau ujung jalan sedangkan sisi menyatakan ruas-ruas jalan yang dapat dilalui. Asumsi dari penyusunan peta adalah jalan selalu dapat dilalui dari dua arah serta setiap ruas jalan memiliki bobot yang menyatakan jarak antar simpul (dalam km). Jarak dari dua buah simpul dihitung dengan menggunakan rumus haversine.

Algoritma A*

Algoritma A* merupakan salah satu algoritma yang populer untuk mencari lintasan terpendek antara dua titik. Pada prinsipnya, A* bekerja dengan cara Greedy Best-First-Search yang memanfaatkan sebuah heuristik dalam mencari sebuah lintasan. Pada proses pencarinya, Algoritma A* menggabungkan apa yang dilakukan Algoritma Dijkstra (memprioritaskan simpul yang lebih dekat dengan titik awal) dengan Greedy Best-First-Search (memprioritaskan simpul yang lebih dekat dengan titik akhir). Dalam terminologi standar, Algoritma A* dinyatakan dengan rumus

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

dengan $g(n)$ merepresentasikan *exact cost* yang dibutuhkan dari *starting node* ke titik n, sedangkan $h(n)$ merepresentasikan nilai estimasi heuristik dari titik n ke *goal node*.

Deskripsi Langkah-Langkah

Langkah-langkah program :

1. Baca persoalan
2. Tambahkan start node ke *opened list*
3. Selama *opened list* tidak kosong, lakukan :
 - a. Cari f dengan nilai terkecil yang ada di *opened list* yang akan disebut n (*current node*)
 - b. Untuk setiap node yang bertetangga dengan *current node*, lakukan :
 - i. Jika simpul hidup belum berada di *opened list*, tambahkan node tersebut ke dalam list, kemudian jadikan *current node* sebagai *parent* dari *node tetangga*, hitung *cost* dari *node tetangga* yg ditambahkan.
 - ii. Jika simpul hidup sudah berada di *opened list*, lakukan pengecekan apakah path yang dibentuk lebih baik dari segi cost. Jika ada assign sebagai *current node*.
 - c. Berhenti ketika :
 - i. Path telah ditemukan, atau
 - ii. Gagal menemukan node tujuan dan *opened listnya* kosong yang berarti tidak ada path.

Source Code

```
import pandas as pd
import gmaps
import gmaps.datasets
from math import radians, cos, sin, asin, sqrt

gmaps.configure(api_key='AIzaSyBlvE6HXrmuztPHa5sa6JIKXraPGrG1Bcc') #Kalo udah
expire, kontak kita ya kak hehe :D
```

```
class Node:
    def __init__(self, nPred, location, next, trail):
        self.nPred = nPred #int
        self.location = location #location
        self.next = next #Node
        self.trail = trail #SuccNode

class SuccNode:
    def __init__(self, succ, nextT, weight):
        self.succ = succ #Node
        self.nextT = nextT #SuccNode
        self.weight = weight #float

class Graph:
    def __init__(self, First):
        self.First = First #Node

    def SearchNode(self, loc):
        P = self.First
        while (P != None and (P.location.name != loc.name)):
            P = P.next
        return P

    def SearchEdge(self, prec, succ): #SuccNode
        P = self.SearchNode(prec)
        if (P == None):
            return None
        T = P.trail
        if (T == None):
            return None
        while (T.succ.location.name != succ and T.nextT != None):
            T = T.nextT
        if (T.succ.location.name != succ):
            return None
```

```

    return T

def InsertNode(self, location):
    Last = self.First

    P = Node(0, location, None, None)
    if (P != None):
        while (Last.next != None) :
            Last = Last.next
        Last.next = P

def InsertEdge(self, source, destination, weight):
    Pprec = self.SearchNode(source)
    Psucc = self.SearchNode(destination)

    if (self.SearchEdge(source, destination) == None):
        T = Pprec.trail

        if (T == None):
            temp = SuccNode(Psucc, None, weight)
            Pprec.trail = temp
        else:
            while (T.nextT != None):
                T = T.nextT
            temp = SuccNode(Psucc, None, weight)
            T.nextT = temp
            Psucc.nPred += 1

    Pprec = self.SearchNode(destination)
    Psucc = self.SearchNode(source)

    if (self.SearchEdge(destination, source) == None):
        T = Pprec.trail

        if (T == None):
            temp = SuccNode(Psucc, None, weight)
            Pprec.trail = temp
        else:
            while (T.nextT != None):
                T = T.nextT
            temp = SuccNode(Psucc, None, weight)
            T.nextT = temp
            Psucc.nPred += 1

class Location:
    def __init__(self, longitude, latitude, name):
        self.longitude = longitude

```

```
    self.latitude = latitude
    self.name = name
```

```
#Fungsi heuristik untuk menghitung perkiraan jarak antar dua titik
def h(loc1, loc2):
    #konversi nilai koordinat menjadi radian
    lon1 = radians(loc1.latitude)
    lon2 = radians(loc2.latitude)
    lat1 = radians(loc1.longitude)
    lat2 = radians(loc2.longitude)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

    c = 2 * asin(sqrt(a))

    # Jari-jari bumi (km)
    r = 6371

    # result
    return(c * r)
```

```
#Menerima sebuah graph (tidak harus fully connected), suatu start dan end node
#Mengembalikan tuple of path dan shortest distance antara start dan end node bila
ditemukan, dan None sebaliknya
def AStar(graph, start, end):
    opened = [start] #list node hidup (yang sedang dibangkitkan)
    closed = [] #list node mati
    g = {} #jarak terpendek (sementara) start node ke setiap node dalam graph
    parents = {} #node sebelum current node
    distance = 0 #jarak terpendek start node dan end node

    g[start] = 0 #jarak start node ke start node = 0

    parents[start] = start #inisiasi parent dari start node = diri sendiri

    while (len(opened) > 0): #masih ada simpul hidup
        n = None #inisiasi current node

        for alive in opened: #simpul ekspan
            #cek apakah ada node hidup lain yang memiliki cost lebih rendah dari n
```

```

if (n == None or g[alive] + h(alive.location, end.location) < g[n] +
    h(n.location, end.location)):
    n = alive

if (n == end or graph.SearchNode(n.location) == None): #akhir
    pass
else:
    p = n.trail #inisiasi trail node
    while (p != None):
        if (p.succ not in opened and p.succ not in closed):
            opened.append(p.succ) #menambahkan succ node p ke opened list
            jika belum pernah dicek
                parents[p.succ] = n #assign current node sebagai parent dari
            succ node
                g[p.succ] = g[n] + p.weight #assign jarak dari succ node ke
            start
            else:
                if (g[p.succ] > g[n] + p.weight): #cek apakah succ node nilai
            fungsi g yang lebih baik
                    g[p.succ] = g[n] + p.weight
                    parents[p.succ] = n

                if (p.succ in closed): #remove dari closed jika sebelumnya
            sudah dimatikan
                    closed.remove(p.succ)
                    opened.append(p.succ)
            p = p.nextT

if (n == None): #tidak ada jalan
    print("Gaada jalan")
    return

if (n == end): #sampai pada node tujuan
    path = [] #inisiasi path

    while (parents[n] != n): #backtrack end node ke start node
        path.append(n)
        n = parents[n]

    path.append(start) #menambahkan start node
    path.reverse() #reverse list

    #print path
    print("Path found: ")
    for node in path:
        print(node.location.name, end=" ")
    print()

```

```

#accumulated distance
for i in range(len(path) - 1):
    distance += h(path[i].location, path[i+1].location)
return (path, distance)
return

#terminasi
opened.remove(n)
closed.append(n)

print("Gaada jalan")
return

```

```

# Open file
namafile = input("Masukkan nama file: ")
file1 = open(namafile, 'r')
counter = 0
nodes = 0
locations = []
adj_matrix = []

# Membentuk graph dari file eksternal
for line in file1:
    if (counter == 0):
        #Initiate jumlah node
        nodes = int(line.strip())
    elif (counter <= nodes):
        c_line = line.split()
        c_loc = Location(float(c_line[1]), float(c_line[2]), c_line[0]) #form
location from read line
        if (counter == 1):
            #Initiate graph
            start_node = Node(0, c_loc, None, None)
            graph = Graph(start_node)
        else:
            #Insert node
            graph.InsertNode(c_loc)
            locations.append(c_loc)
    else: #counter > nodes
        c_line = line.split()
        adj_matrix.append(c_line)
    counter += 1

#Insert edges

```

```

for i in range(nodes):
    for j in range(nodes):
        if (adj_matrix[i][j] == '1'):
            graph.InsertEdge(locations[i], locations[j], h(locations[i],
locations[j]))

print("Successfully constructed graph")

# Close file
file1.close()

```

```

#DRAW GRAPH
fig = gmaps.figure()
markerList = []
infoList = []
lineList = []

#Text box
info_box_template = """
<dl>
<dt>{0}</dt>
</dl>
"""

i = 1
cNode = graph.First
while (cNode != None):
    #Node
    cNodeLoc = (cNode.location.longitude, cNode.location.latitude)
    markerList.append(cNodeLoc)
    infoList.append(info_box_template.format(cNode.location.name))

    #Trail Nodes
    tNode = cNode.trail
    while (tNode != None):
        tNodeLoc = (tNode.succ.location.longitude, tNode.succ.location.latitude)
        lineList.append(gmaps.Line(start=cNodeLoc, end=tNodeLoc,
stroke_weight=3.0))
        tNode = tNode.nextT

    #Next Node
    cNode = cNode.next
    i += 1

#Draw figure and markers

```

```
markers = gmmaps.marker_layer(markerList, info_box_content=infoList)
drawing = gmmaps.drawing_layer(features=lineList)
fig.add_layer(markers)
fig.add_layer(drawing)
fig
```

```
#RESULT
markerList = []

start_loc_name = input("Masukkan nama start node: ")
end_loc_name = input("Masukkan nama end node: ")
start_loc = Location(None, None, start_loc_name)
end_loc = Location(None, None, end_loc_name)

#Inisiasi start dan end node
Start = graph.SearchNode(start_loc)
End = graph.SearchNode(end_loc)
markerList.append((Start.location.longitude, Start.location.latitude))
markerList.append((End.location.longitude, End.location.latitude))

#Menjalankan algoritma A*
a_star_result = AStar(graph, Start, End)
if (a_star_result != None):
    path = a_star_result[0]
    distance = a_star_result[1]
    print("distance: {0:.4f}km" .format(distance))

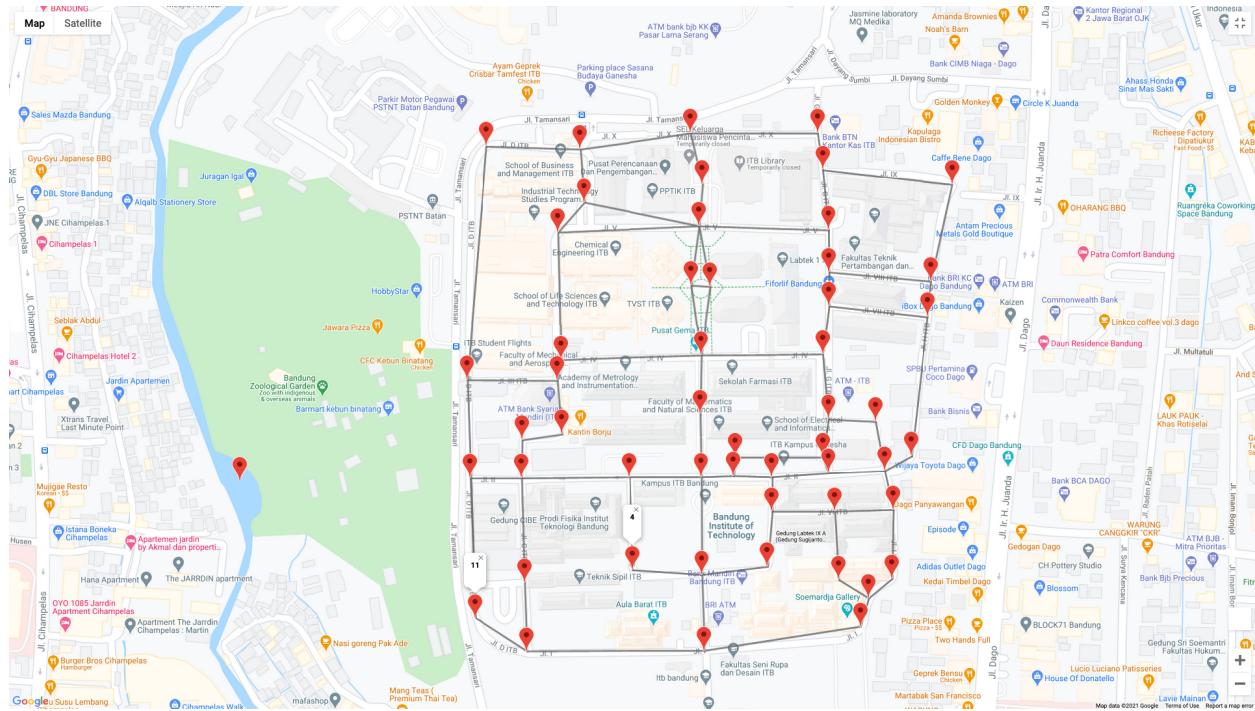
#draw result map
fig2 = gmmaps.figure()
if (a_star_result != None):
    for i in range(len(path)-1):
        a = (path[i].location.longitude, path[i].location.latitude)
        b = (path[i+1].location.longitude, path[i+1].location.latitude)
        temp = gmmaps.directions_layer(a, b, show_markers=False,
travel_mode='WALKING')
        fig2.add_layer(temp)

    markers = gmmaps.marker_layer(markerList)
    fig2.add_layer(markers)
fig2
```

Hasil Screenshot

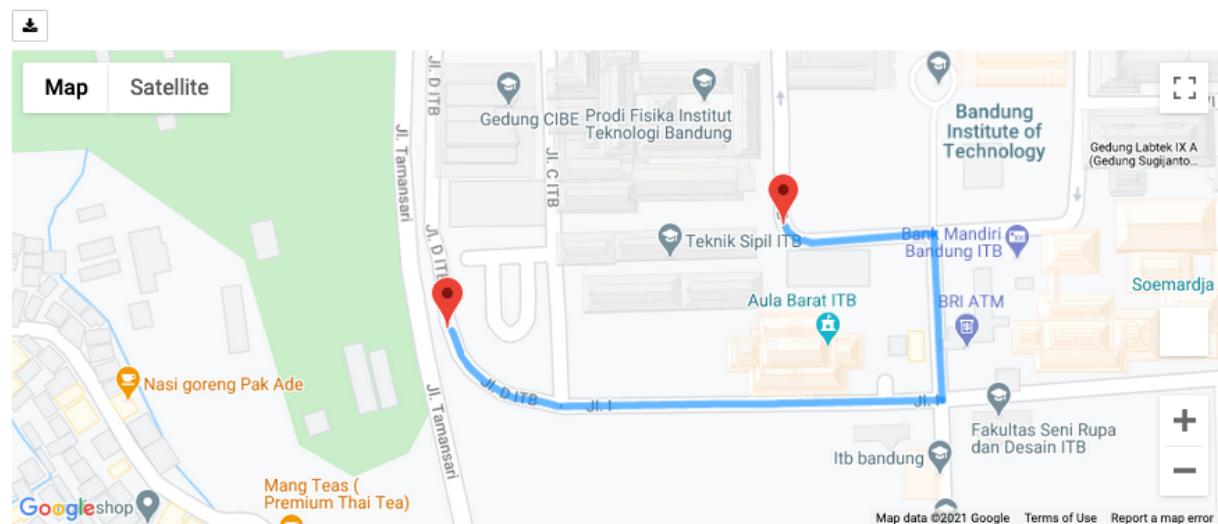
Test Case 1: Peta ITB

File: itb.txt



Gambar 1. Ilustrasi peta ITB dalam graph

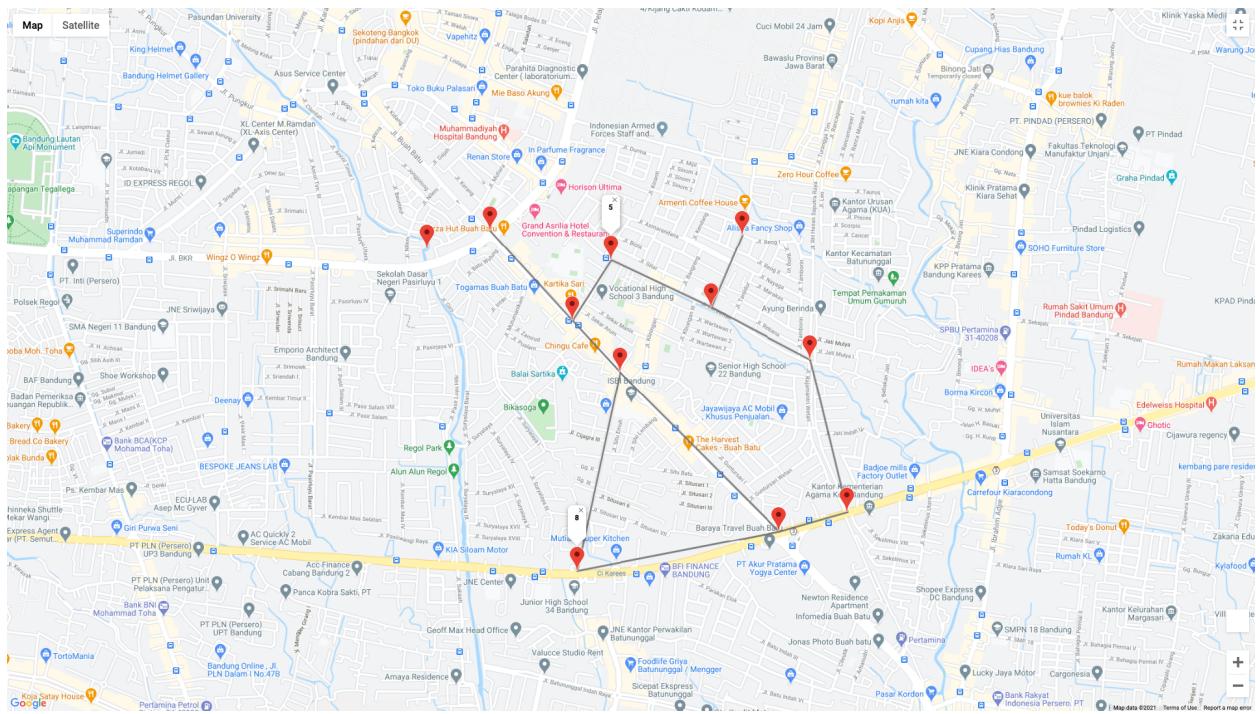
```
Masukkan nama start node: 4
Masukkan nama end node: 11
Path found:
4 3 1 2 11
Distance: 0.3963km
```



Gambar 2. Rute hasil prediksi algoritma A*

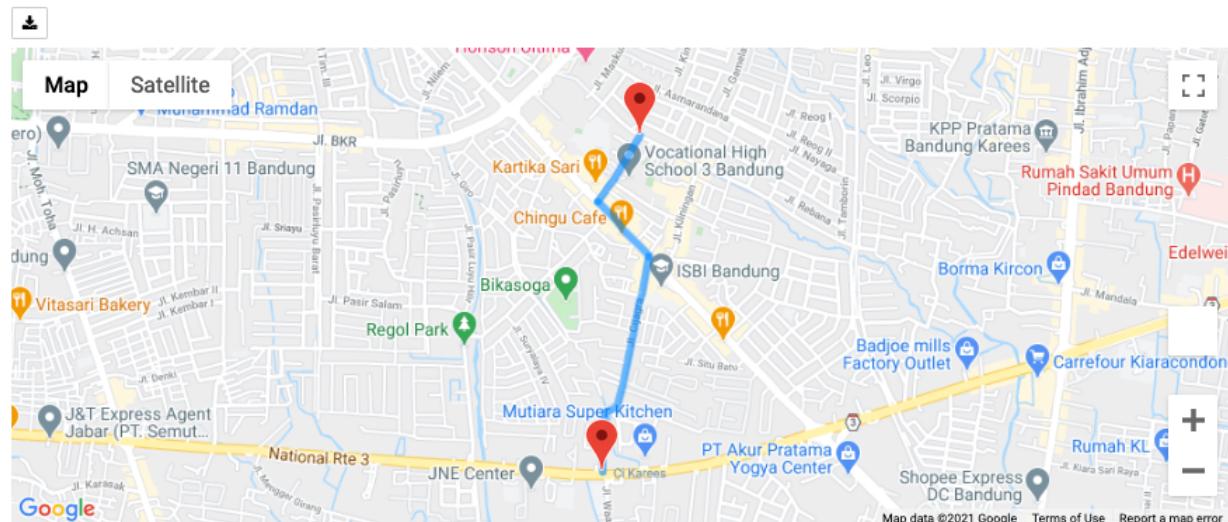
Test Case 2: Peta Buah Batu

File: buahbatu.txt



Gambar 3. Ilustrasi peta Buah Batu dalam graph

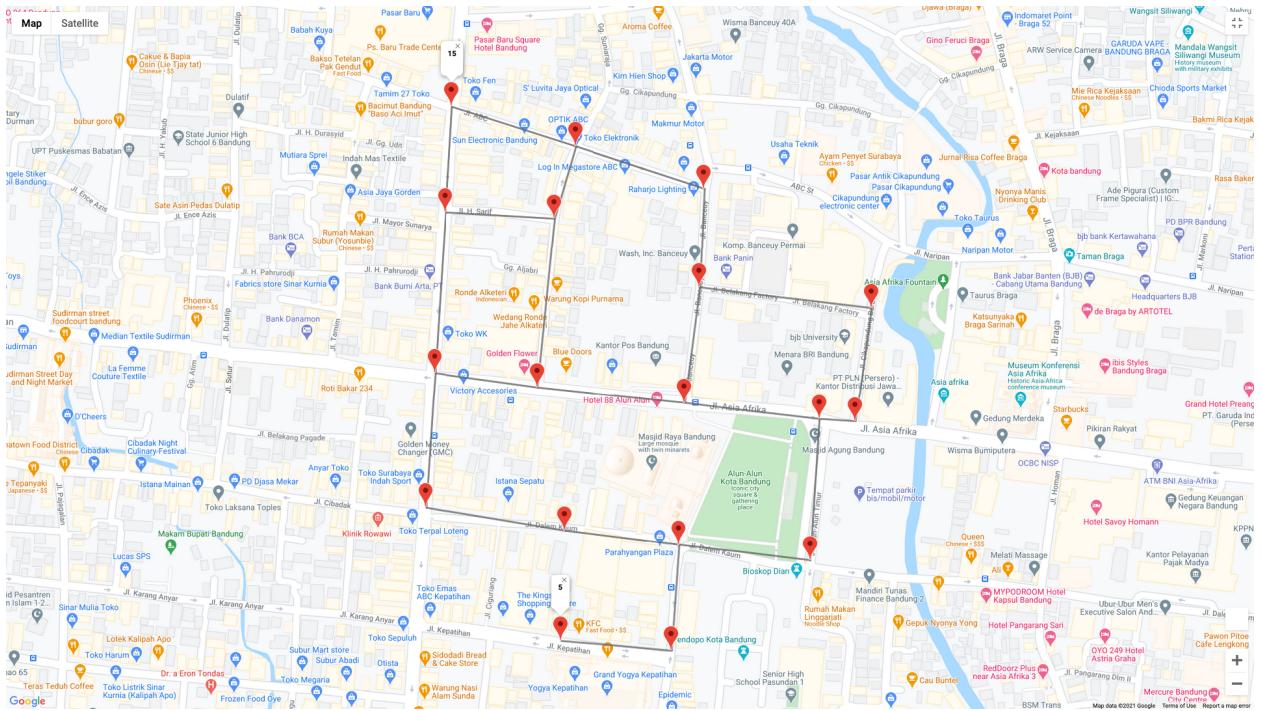
```
Masukkan nama start node: 8  
Masukkan nama end node: 5  
Path found:  
8 7 6 5  
Distance: 1.4244km
```



Gambar 4. Rute hasil prediksi algoritma A*

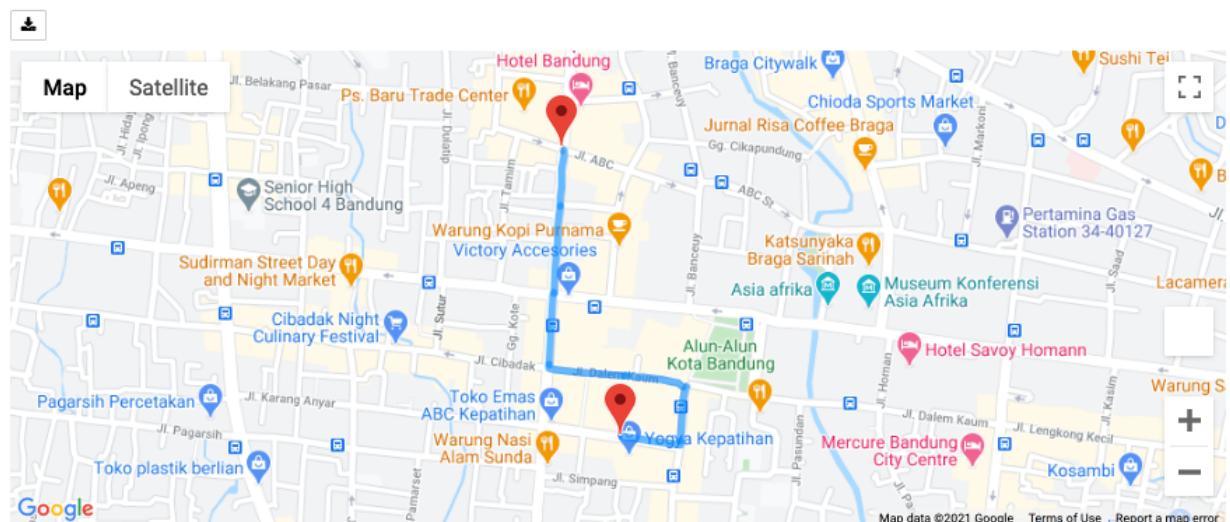
Test Case 3 : Peta Alun-alun

File : alunalun.txt



Gambar 5. Ilustrasi peta Alun-alun dalam graph

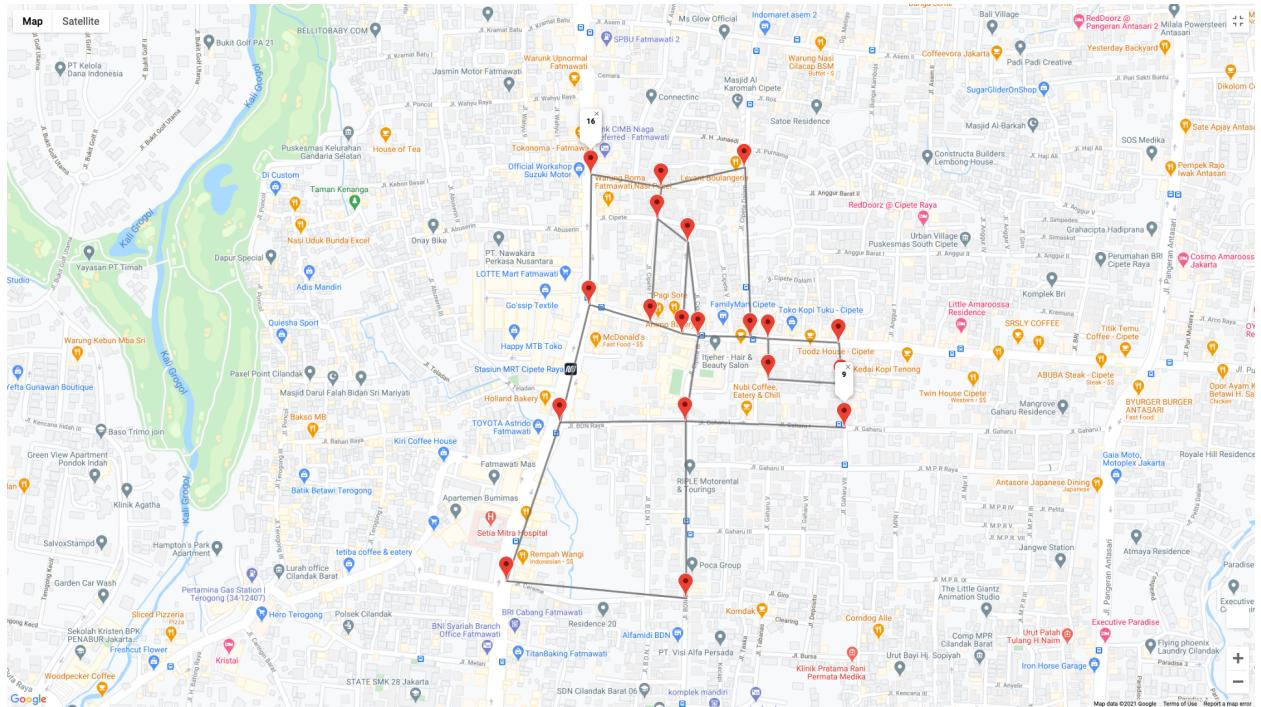
```
Masukkan nama start node: 5
Masukkan nama end node: 15
Path found:
5 6 7 3 1 2 14 15
Distance: 0.9026km
```



Gambar 6. Rute hasil prediksi algoritma A*

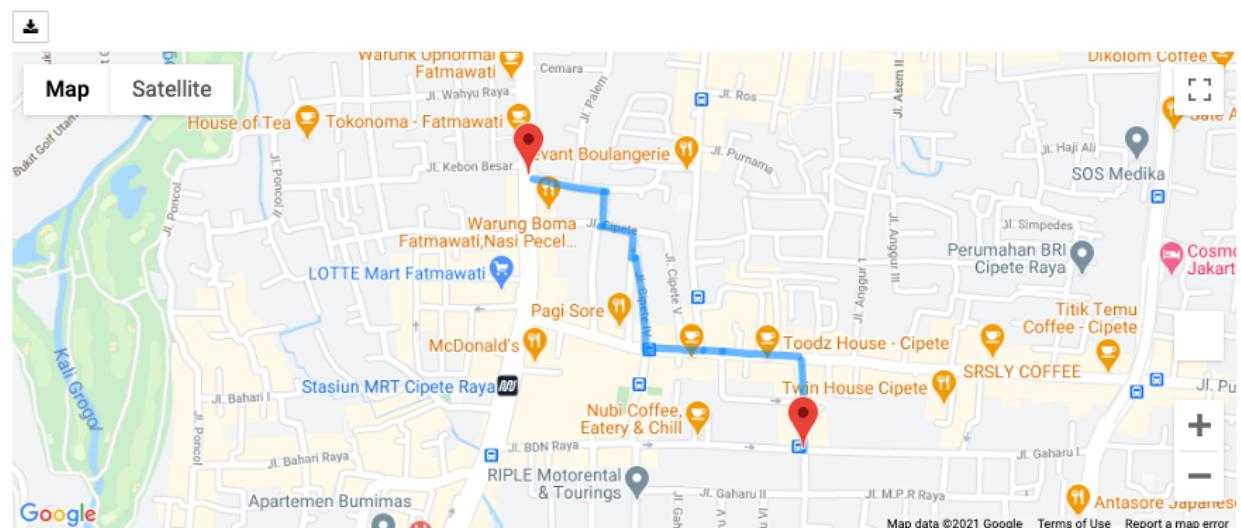
Test Case 4 : Daerah Sekitar Tempat Tinggal - Cilandak

File : deketrumah.txt



Gambar 7. Ilustrasi peta Cilandak dalam graph

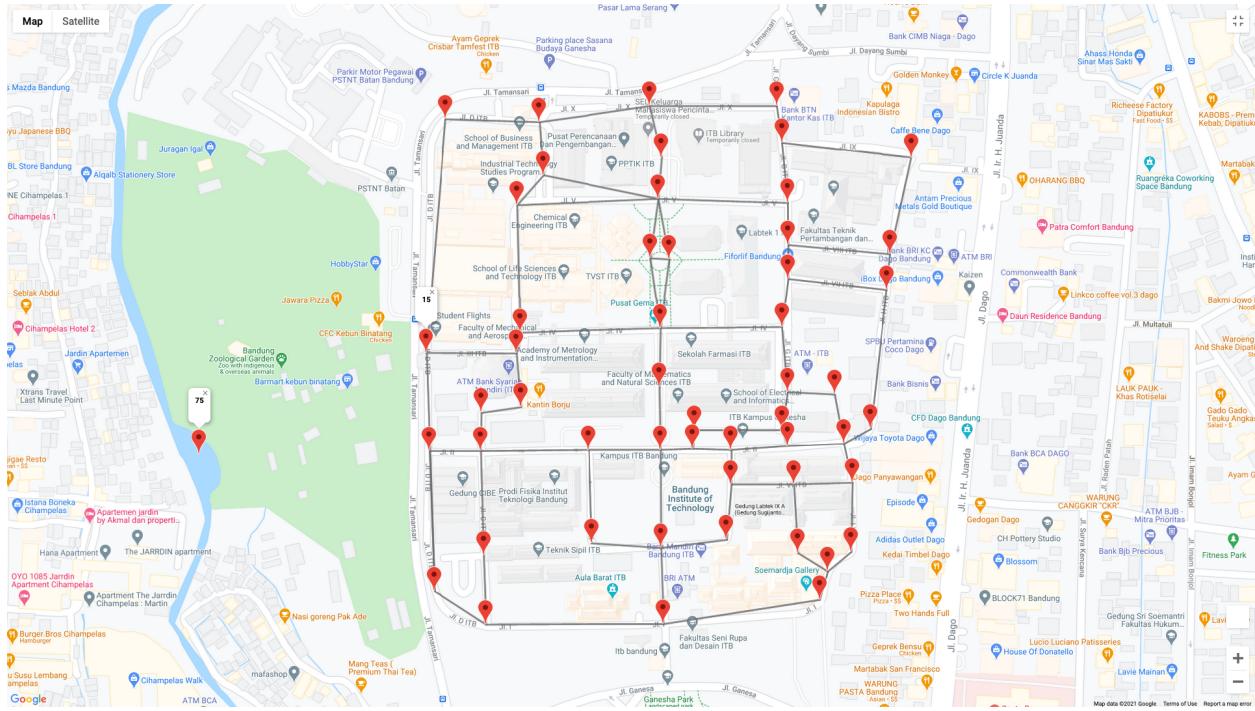
```
Masukkan nama start node: 9
Masukkan nama end node: 16
Path found:
9 8 6 5 13 3 11 17 18 16
Distance: 0.9527km
```



Gambar 8. Rute hasil prediksi algoritma A*

Test Case 5 : Daerah Terisolasi di Sekitar ITB

File : itb.txt

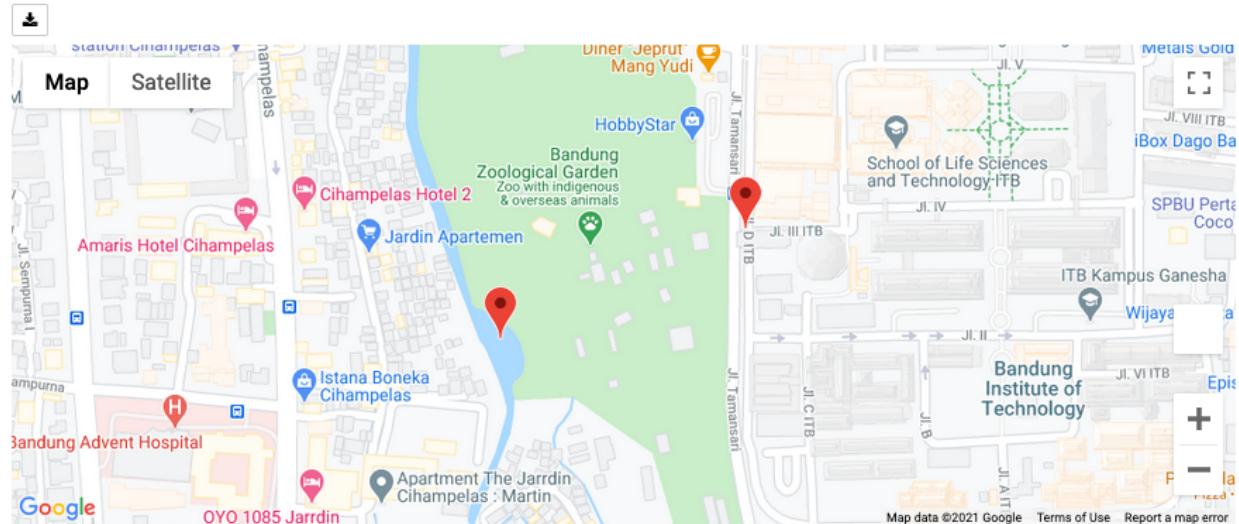


Gambar 9. Ilustrasi peta ITB dalam graph

Masukkan nama start node: 75

Masukkan nama end node: 15

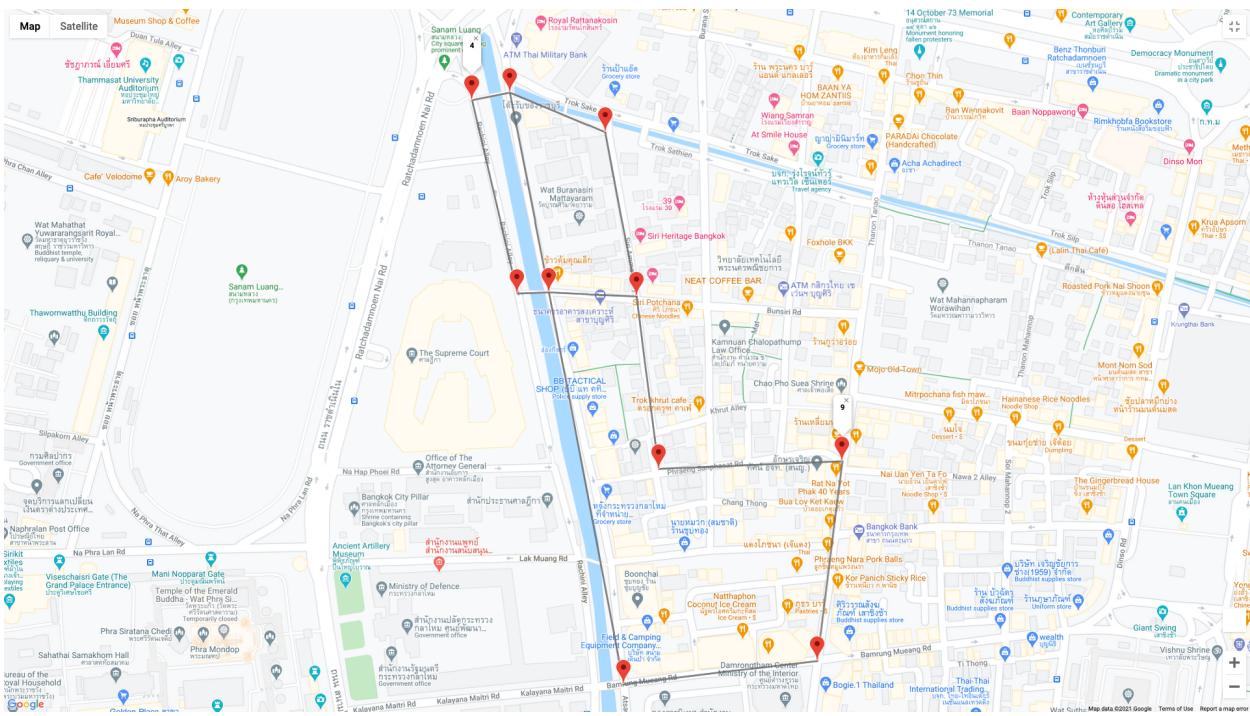
Gaada jalan



Gambar 10. Ilustrasi kasus node tidak terhubung

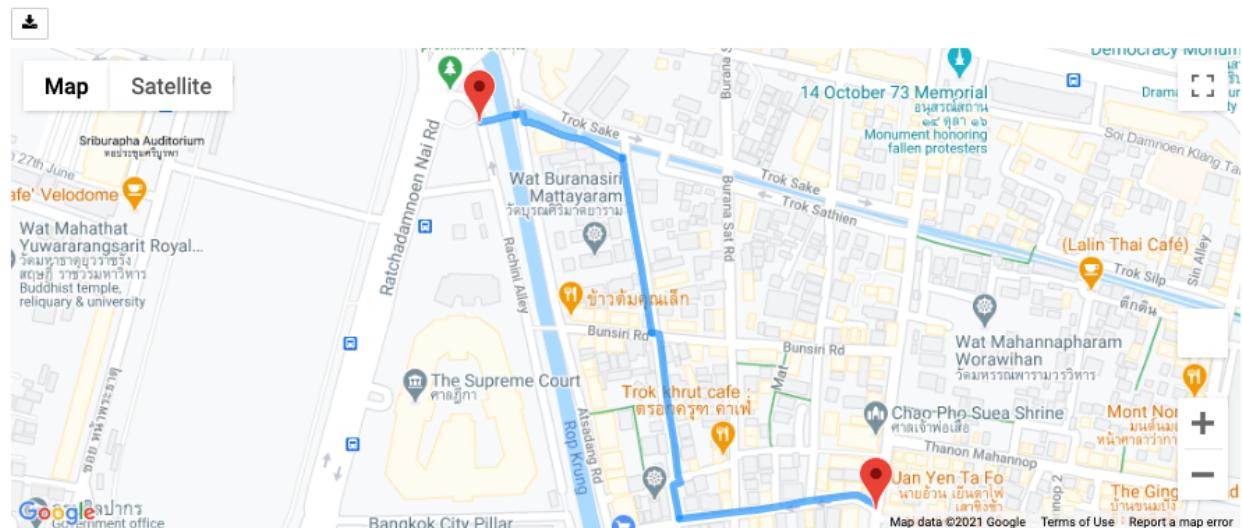
Test Case 6 : Map Bangkok

File : bangkok.txt



Gambar 11. Ilustrasi peta Bangkok dalam graph

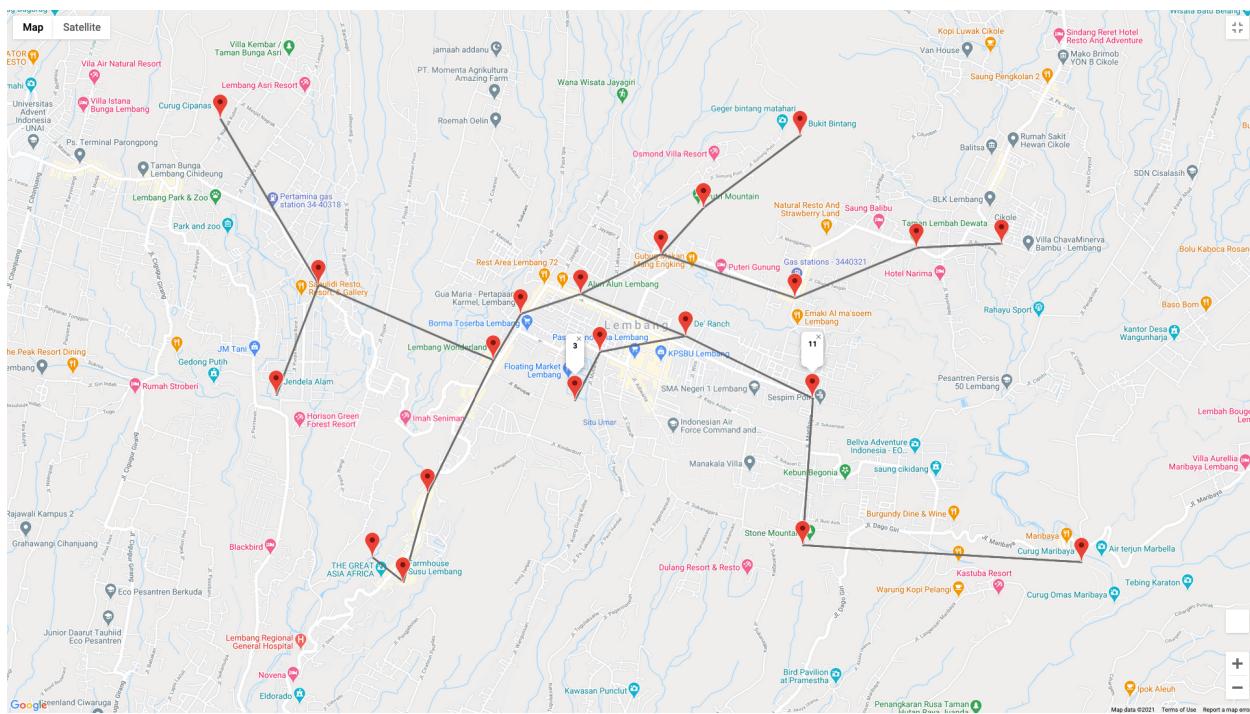
```
Masukkan nama start node: 4
Masukkan nama end node: 9
Path found:
4 5 6 8 10 9
Distance: 0.6629km
```



Gambar 12. Rute hasil prediksi algoritma A*

Test Case 7 : Map Lembang

File : lembang.txt



Gambar 13. Ilustrasi peta Lembang dalam graph

```
Masukkan nama start node: 3
Masukkan nama end node: 11
Path found:
3 20 5 11
Distance: 2.3004km
```



Gambar 14. Rute hasil prediksi algoritma A*

Checklist

Poin	Ya	Tidak
1. Program dapat menerima input graf	√	
2. Program dapat menghitung lintasan terpendek	√	
3. Program dapat menampilkan lintasan terpendek serta jaraknya	√	
4. Program dapat menerima input peta dengan Google Map API dan menampilkan peta	√	

Link File

Github: <https://github.com/salsabiilashifa11/AStarPathFinder>