

LAPORAN TUGAS KECIL 3

STRATEGI ALGORITMA - IF2211

“Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek”



Dibuat Oleh :

Mohammad Sheva Almeyda Sofjan - 13519018 K01

Alif Bhadrika Parikesit - 13519186 K04

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

DESKRIPSI PERSOALAN

Algoritma 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 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.

Akan dibuat program dengan spesifikasi :

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.



ALGORITMA A*

A* (A-Star) merupakan algoritma pencarian path/lintasan terpendek dari suatu simpul start ke simpul tujuan pada suatu graf, dengan ide untuk mencari path ke simpul tujuan yang memiliki cost(jarak tempuh) seminimum mungkin dengan menghindari ekspansi path yang memiliki estimasi cost yang tidak minimum.

Cost pada algoritma A* diestimasi menggunakan fungsi $f(n) = g(n) + h(n)$, dimana n merupakan simpul selanjutnya pada path, $g(n)$ merupakan jarak tempuh dari simpul start ke simpul n, $h(n)$ merupakan jarak heuristik (euclidean/ haversian/ straight-line distance) dari simpul n ke simpul tujuan, dan $f(n)$ merupakan estimasi total cost (total jarak) yang harus diminimalisasi dalam pencarian path.

Langkah-langkah pada Algoritma A* :

1. Mula-mula tetapkan simpul start dan simpul tujuan, inisialisasi nilai $f = g + h$ awal
2. Ekspansi simpul start (secara breadth-first/melebar)
3. Hitung nilai $f(n)+g(n) = h(n)$ tiap simpul yang telah diekspansi
4. Ekspansi simpul yang belum dikunjungi dengan $f(n)$ minimum
5. Ulangi langkah 3-4 hingga simpul tujuan dikunjungi (dengan $f(n)$ minimum) atau hingga dapat disimpulkan tidak terdapat path dari simpul start ke simpul tujuan

Pada program ini, alur jalan program adalah sebagai berikut :

1 Mula-mula program akan menerima input graf peta dalam file berekstensi .txt dengan format :

Baris pertama merupakan jumlah simpul pada graf (n)

Baris kedua hingga $n+1$ berisi nama simpul diikuti dengan koordinat latitude dan longitude dipisahkan dengan spasi

n baris sisanya berisi matriks ketetanggaan (*adjacency matrix*) berukuran $n \times n$

Sebagai contoh file input sebagai berikut :

tangerang.txt

```
9
LapAhYani -6.171034876040317 106.63405218570935
SgMogot -6.169202277962933 106.63341102790366
MCD -6.171444092005021 106.63140107694329
SgMogotKs -6.175586907344772 106.62993055181754
SgKsDamyati -6.177613423067104 106.6299920128765
GOR -6.178159025211438 106.63309637679686
Stasiun -6.176801597885472 106.63273056128227
SgSolehAliDamyati -6.177405618504051 106.6346512135528
SgSolehAliAhYani -6.170889811884585 106.63519363668155
0 1 0 0 0 0 0 1
1 0 1 1 0 0 0 0
0 1 0 1 0 0 0 0
0 1 1 0 1 0 1 0
0 0 0 1 0 1 1 0
0 0 0 0 1 0 0 1 0
0 0 0 1 1 0 0 1 0
0 0 0 0 1 1 1 0 1
1 0 0 0 0 0 1 0
```

2. Lalu program akan menampilkan *pop-up* graf
3. Tutup *pop-up* graf, lalu program menerima input simpul awal dan simpul tujuan, dan memproses jarak antara simpul awal dan simpul tujuan menggunakan algoritma A*
4. Pada akhirnya program akan menampilkan lintasan, jarak, serta *pop-up* graf dengan pewarnaan simpul dan sisi graf pada lintasan hasil proses algoritma A* jika terdapat solusi, atau pesan keterangan tidak terdapat lintasan dari simpul awal ke simpul tujuan

KODE PROGRAM

Program dibuat menggunakan bahasa pemrograman *Python* (*Python 3*) dengan *interface* berupa command line untuk *input command* dan *stream output*. GUI dipakai untuk *output* berupa visualisasi graf (dengan bantuan library *pyplot*).

```
# import library and external utility tools
import math
import networkx as nx
import matplotlib.pyplot as plt
import sys
```

```
class Vertex:
    def __init__(self, _name, _lat, _long):
        self.name = _name
        self.lat = _lat # Latitude
        self.long = _long # Longitude
        self.coorX = 6371 * math.cos(_lat) * math.cos(_long) #cartesian x coordinate
        self.coorY = 6371 * math.cos(_lat) * math.sin(_long) #cartesian y coordinate
        self.f = 0 #Estimated total cost of path from self to dst
        self.g = 0 # cost so far to reach self
        self.h = 0 # estimated cost from self to goal
        self.parent = None

    def printInfo(self):
        print(self.name,"Coordinate : (",self.lat,", ",self.long,) : Cartesian
        (",self.coorX,", ",self.coorY)
```

```
class Graph:
    def __init__(self, _numVertices):
        ...
        Constructor
        ...
        self.numVertices = _numVertices
        self.vertices = []
        self.adj = [[0 for i in range(_numVertices)] for j in range(_numVertices)]

    def addVertex(self, _name, _lat, _long):
        ...
        Vertex Adder
        ...
        newVertex = Vertex(_name, _lat, _long)
        self.vertices.append(newVertex)

    def addEdge(self, v1, v2):
        ...
        Edge adder
        ...
        idx1 = self.findVertexIdx(v1)
        idx2 = self.findVertexIdx(v2)
        self.adj[idx1][idx2] = self.calcDist(v1,v2)

    def syncAdj(self,adjmat):
```

```

    ...
    update adj matrix
    ...
    self.adj = adjmat

def findVertexByIdx(self, idx):
    ...
    search vertex by idx , return its name
    ...
    return self.vertices[idx].name

def findVertexIdx(self, _name):
    ...
    search vertex idx by name, return its idx
    ...
    for i in range (self.numVertices):
        if (self.vertices[i].name == _name):
            return i
    return -1

def findIdxByVertex(self,V):
    ...
    search vertex idx by vertex, return its idx
    ...
    for i in range(self.numVertices):
        if(self.vertices[i].name == V.name):
            return i
    return -1
def printGraph(self):
    ...
    print graph
    ...
    for i in range(self.numVertices):
        self.vertices[i].printInfo()

def calcDist(self,srcName,dstName):
    ...
    passer to haverDist
    ...
    src = self.vertices[self.findVertexIdx(srcName)]
    dst = self.vertices[self.findVertexIdx(dstName)]
    return self.haverDist(src,dst)

def haverDist(self,src,dst):
    ...
    in m , haversine, calculate heuristic distance between two points
    ...
    lat1 = math.radians(src.lat)
    lat2 = math.radians(dst.lat)
    long1 = math.radians(src.long)
    long2 = math.radians(dst.long)
    deltaLat = lat2 - lat1
    deltaLong= long2 - long1
    a = (math.sin(deltaLat/2))**2 +
        math.cos(lat1)*math.cos(lat2)*(math.sin(deltaLong/2))**2
    c = 2 * math.asin(math.sqrt(a))
    return 6371 * c * 1000

```

```

def generateSucc(self,current):
    """
    Generate successor(neighbor) node of current node
    """
    succNode = []
    for i in range(self.numVertices):
        if(self.adj[self.findIndexByVertex(current)][i] and
           i!=self.findIndexByVertex(current)):
            succNode.append(self.vertices[i])
    return succNode

def computeAStar(self,srcName,dstName):
    """
    Compute Distance between two nodes using A* Algo
    """
    if(self.findVertexIdx(srcName)==-1 or self.findVertexIdx(dstName)==-1 ):
        # Exit if node not found
        return []
    src = self.vertices[self.findVertexIdx(srcName)] # src node
    dst = self.vertices[self.findVertexIdx(dstName)] # dst node

    openList =[] # visited node + not expanded (queue node)
    closedList =[] # visited + expanded node
    for i in range(self.numVertices): # init h distance
        self.vertices[i].h = self.haverDist(self.vertices[i],dst)
    src.f = src.h
    openList.append(src) # init
    emp = False # openlist emptiness indicator
    while(not emp):
        for i in range(self.numVertices):
            # Calculate f estimation in each iteration / hitung nilai f di tiap iterasi
            self.vertices[i].f = self.vertices[i].g + self.vertices[i].h
        currIdx = minFIdx(openList)
        currNode = openList[currIdx]
        # currentNode is node in the openList which f value is lowest / node yang nilai f nya terendah

        if(currNode == dst): # found dest node, return result
            pathList = []
            currentNode = currNode
            while currentNode is not None:
                pathList.append(currentNode.name)
                currentNode = currentNode.parent
            return currNode.f,pathList[::-1]

        openList.remove(currNode) # remove currNode from openList (queue)
        succList = self.generateSucc(currNode) # Generate successor node list
        for succNode in succList: # foreach successor node
            tempCost = currNode.g + self.haverDist(currNode,succNode)
            # calculate temporary g value
            if(succNode in openList):
                if(succNode.g <= tempCost):
                    continue
                # continue to next iteration (current successsor node path is least
                # cost path) / ke iterasi selanjutnya, current cost sudah lowestsofar

            elif(succNode in closedList):
                if(succNode.g <= tempCost):

```

```

        continue
        # continue to next iteration (current successor node path is least
        cost path)

        closedList.remove(succNode)
        # else remove from closedlist, add to queue (openlist)

        openList.append(succNode)
        # because current path is not the best one / karena tempCost lebih
        rendah dari current path(succ) g value

    else:
        openList.append(succNode) # not in both, append to queue

        succNode.g = tempCost # sync g value if not continue
        succNode.parent = currNode
        # append currNode to succNode parent (for path later on)

        closedList.append(currNode) # finished exploring currNode
        if(len(openList)==0): # end condition
            emp = True

    if(currNode != dst): # path not found
        return []

def visualize(self, aStarPath = None):
    ...
    for visualizing graph
    ...
    if aStarPath is None:
        path = []
    else:
        path = aStarPath
    Gr = nx.Graph()
    for i in range (self.numVertices): # Initial Node+edge adder
        for j in range (self.numVertices):
            if (i<j):
                if self.vertices[i].name not in Gr.nodes():
                    Gr.add_node(self.vertices[i].name, pos = (self.vertices[i].coorX,
                        self.vertices[i].coorY))
                if self.vertices[j].name not in Gr.nodes():
                    Gr.add_node(self.vertices[j].name, pos = (self.vertices[j].coorX,
                        self.vertices[j].coorY))
                if (self.adj[i][j] != 0):
                    formatted_weight = "{:.3f}".format(self.adj[i][j]/1000)
                    Gr.add_edge(self.vertices[i].name, self.vertices[j].name, weight =
                    formatted_weight, relation = 'notinPath')
            else:
                continue

    if(aStarPath is not None) : # Edge coloring
        for i in range(len(path) - 1):
            f_weight = "{:.3f}".format(self.adj[path[i]][path[i+1]]/1000)
            Gr.add_edge(path[i],path[i+1],weight = f_weight,relation = 'inPath')

    edge_color = {'inPath' : 'red', 'notinPath' : 'blue'}
    node_color = []

```

```

for node in Gr.nodes():
    if node in path:
        node_color.append('red')
    else:
        node_color.append('blue')

weight = nx.get_edge_attributes(Gr, 'weight')
pos = nx.get_node_attributes(Gr, 'pos')
relation = nx.get_edge_attributes(Gr, 'relation')

nx.draw_networkx(Gr, pos, node_color = node_color, edge_color=[edge_color[x] for x in relation.values()])
nx.draw_networkx_edge_labels(Gr, pos, edge_labels = weight)
plt.show()

```

```

# utility function
def parseFile(filename):
    '''
    File Parser
    '''

    try:
        with open('../test/' + filename + ".txt", "r") as file:
            lines = file.readlines()
            # get number of vertices from 1st line
            numVertices = int(lines[0].strip())
            G = Graph(numVertices)

            # construct every vertex from line 2
            for i in range(1, numVertices + 1):
                vertexArgs = lines[i].strip().split(" ")
                G.addVertex(vertexArgs[0], float(vertexArgs[1]), float(vertexArgs[2]))

            # construct every edges from adj matrix
            for i in range(numVertices + 1, len(lines)):
                elmt = lines[i].strip().split(" ")
                rowIdx = i - (numVertices + 1)
                for j in range(numVertices):
                    if elmt[j] == '1':
                        G.addEdge(G.findVertexByIdx(rowIdx), G.findVertexByIdx(j))

    return G
    except:
        print("File not Found. Exiting program . . .")
        sys.exit()

def minFIdx(list):
    '''
    Search minimum f func index from a list
    '''

    min = 0
    for i in range(len(list)):
        if(list[i].f < list[min].f):
            min = i
    return min

```

```

# main program (driver)
import graph
import math
import sys

def start():
    print("#### A* Shortest Path Finder ####")
    filename = input("ENTER MAP NAME: ")
    G = graph.parseFile(filename)

    G.visualize()

    nodes = [node for node in G.vertices]
    print("\nPLACES AT",filename.upper())
    for i in range (len(nodes)):
        print("[",i+1,"]", nodes[i].name)

    src = int(input("SOURCE: "))
    dest = int(input("DESTINATION: "))
    try:
        out = G.computeAStar(nodes[src-1].name,nodes[dest-1].name)
    except:
        print("Node not Found. Exiting program . . .")
        sys.exit()
    if len(out) == 0:
        print("THERE'S NO WAY YOU CAN GO FROM ", nodes[src-1].name," TO "
",nodes[dest-1].name")
        print()
    else:
        print("\nTHE SHORTEST PATH FROM ", nodes[src-1].name," TO ",nodes[dest-1].name)
        for i in range (len(out[1])):
            if i == (len(out[1]) - 1):
                print(out[1][i])
            else:
                print(out[1][i], end=" --> ")
        print("\nWITH DISTANCE {:.3f} KM\n".format(out[0]/1000))
        G.visualize(out[1])

if __name__ == '__main__':
    start()

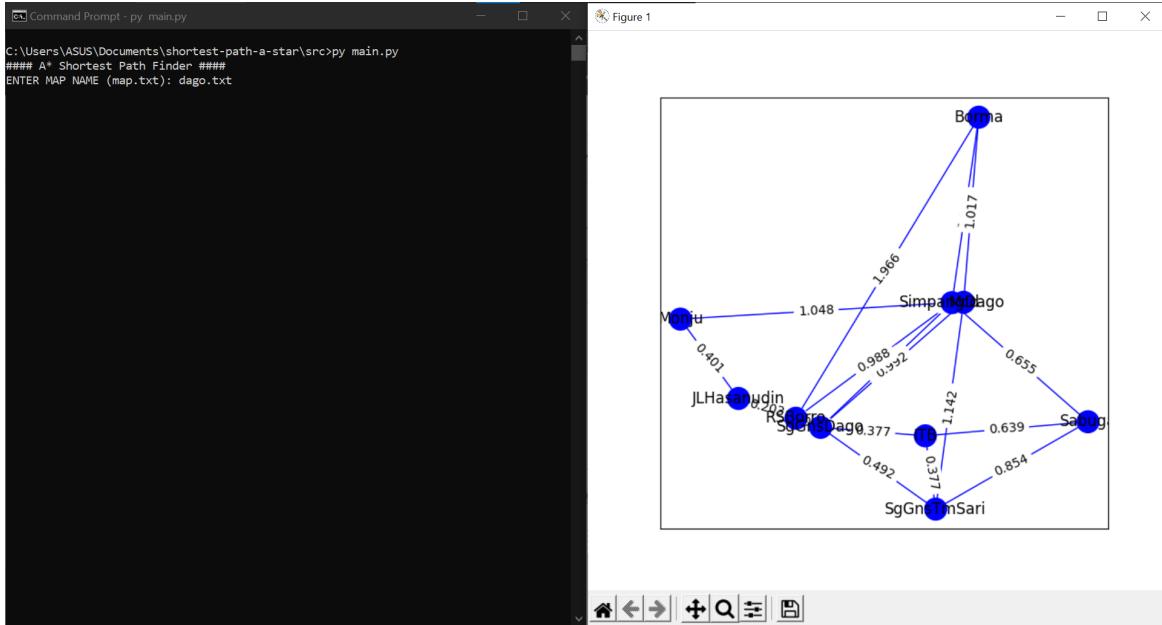
```

SCREENSHOT TESTING DAN PETA

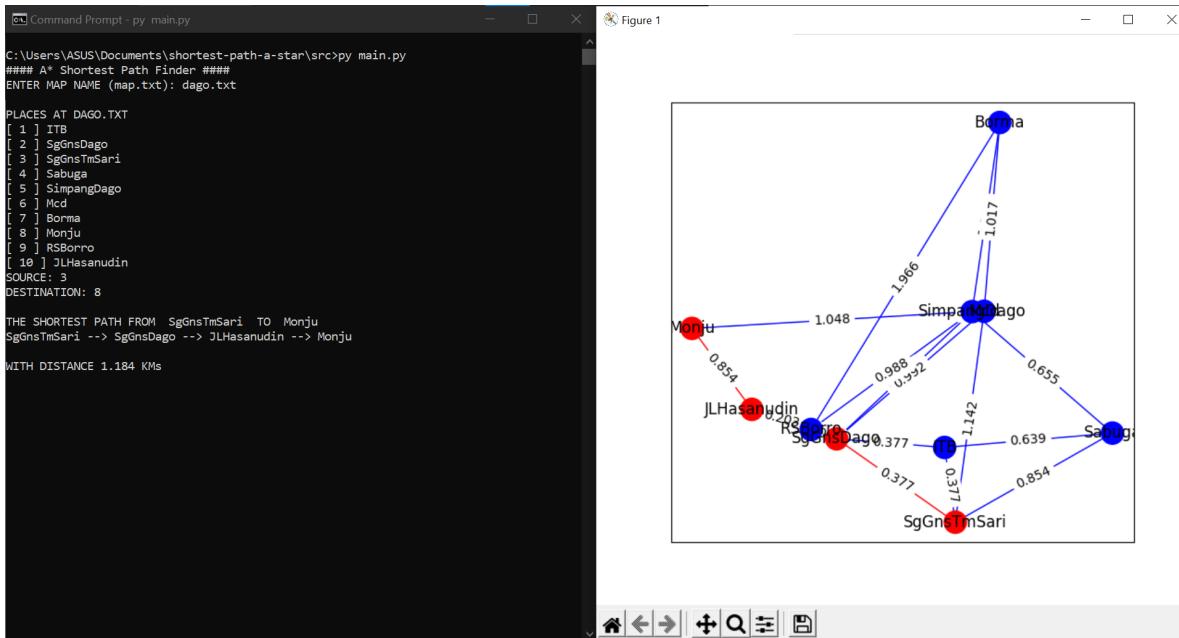
1. dago.txt

```
10
ITB -6.89122436321195 107.61062690985229
SgGnsDago -6.893752115812983 107.61290301289839
SgGnsTmSari -6.893834676957056 107.60844490889858
Sabuga -6.88618206327591 107.60784183643175
SimpangDago -6.885183161339224 107.61368567119648
Mcd -6.884851382107476 107.61346809663375
Borma -6.876948371732721 107.6181023718044
Monju -6.893269219787312 107.6185658449428
RSBorro -6.894064266207075 107.61363267408856
JLHasanudin -6.8948435674046005 107.61529292627816
0 1 1 1 0 0 0 0 0
1 0 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 0 0
1 0 1 0 1 0 0 0 0 0
0 1 0 1 0 1 1 1 1 0
0 1 1 0 1 0 1 0 0 0
0 0 0 0 1 1 0 0 1 0
0 0 0 0 1 0 0 0 0 1
0 1 0 0 1 0 1 0 0 1
0 1 0 0 0 0 0 1 1 0
```

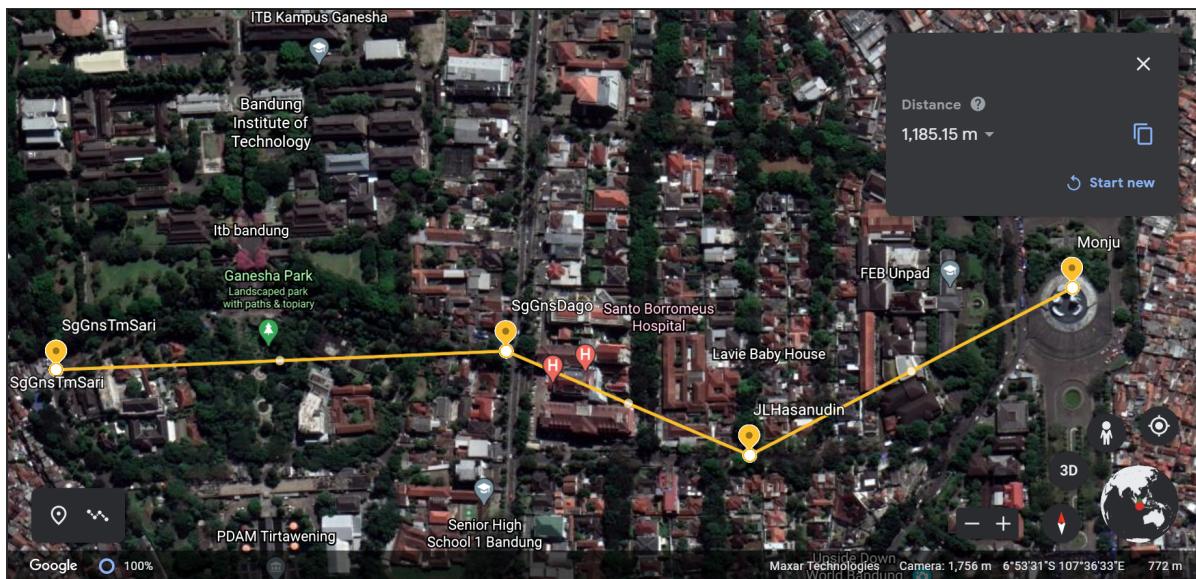
Setelah memasukan nama file external dengan format (namafile.txt), program akan mengeluarkan visualisasi graf peta file external. Bentuk peta dapat tidak rapi dikarenakan posisi node disesuaikan dengan koordinat longitude dan latitude yang dikonversi ke koordinat kartesian yang memungkinkan jarak yang terlalu dekat menyebabkan node hampir berhimpit, contoh SimpangDago dengan Mcd. Label angka di tiap sisi merupakan jarak garis lurus antar node.



Untuk melanjutkan program, *exit* visualisasi graf dan program akan menampilkan daftar node dan meminta masukan source dan destination node yang ingin kita cek. Akan di cek shortest path dari SgGnsTamanSari (Simpang Ganesha-Taman Sari) ke Monju (Monumen Perjuangan). **Disclaimer: shortest path berdasarkan node yang sudah ditambahkan di program dan program tidak menghandle jika ada jalan yang satu arah atau dilarang melintas.**



Hasil keluaran program adalah shortest path dan visualisasinya, serta jarak yang harus ditempuh, yaitu SgGnsTmSari → SgGnsDago → JlHasanudin → Monju, dengan jarak 1.184 km. Berikut ini adalah perbandingan dengan jarak node sebenarnya dengan google earth.



Hasil dapat berbeda karena error manusia dalam memilih titik ujung ruler.

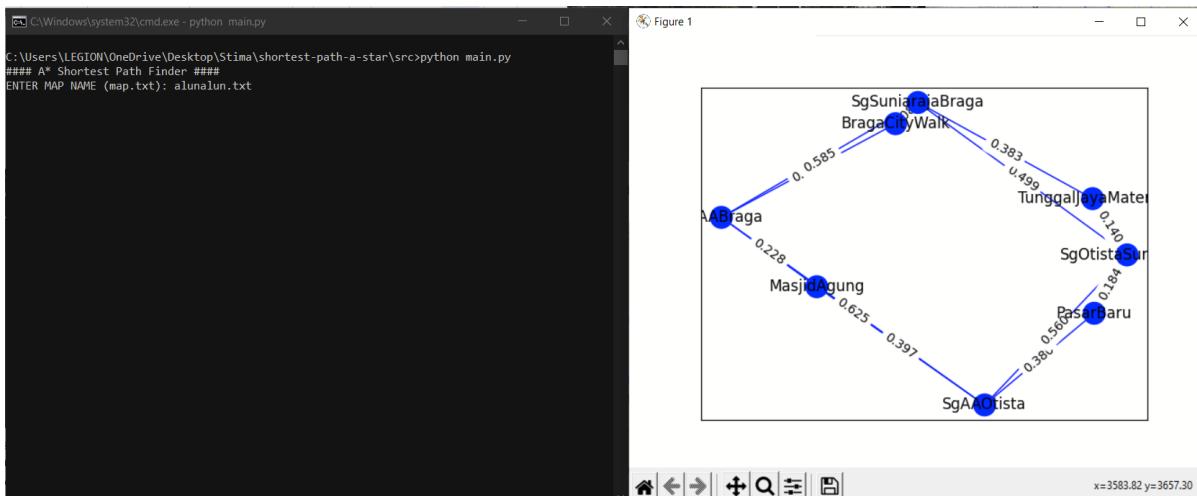
2. alunalun.txt

```

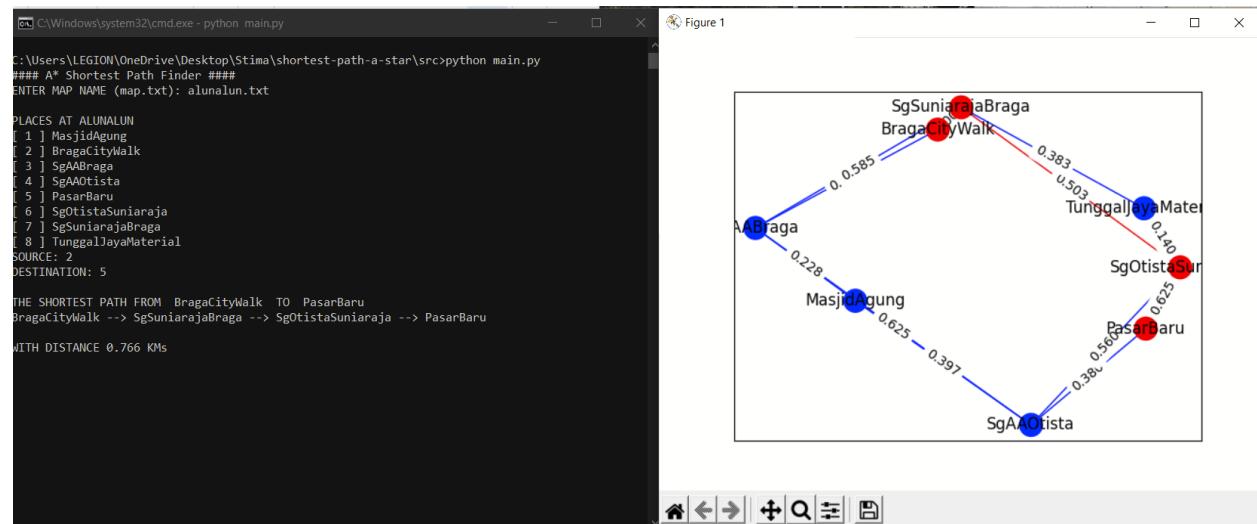
8
MasjidAgung -6.921270227851845 107.60768071813007
BragaCityWalk -6.917007861181857 107.6089578109168
SgAABraga -6.9214680511200815 107.6097393179074
SgAAOtista -6.920839618371926 107.60411261747308
PasarBaru -6.917421664170663 107.60407485109054
SgOtistaSuniaraja -6.915819732408295 107.60448090405069
SgSuniarajaBraga -6.91626174414204 107.60898101914995
TunggalJayaMaterial -6.915314879795135 107.60564379509
0 0 1 1 0 0 0
0 0 1 0 0 0 1 0
1 1 0 1 0 0 1 0
1 0 1 0 1 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 1 0 1 1
0 1 1 0 0 1 0 1
0 0 0 0 0 1 1 0

```

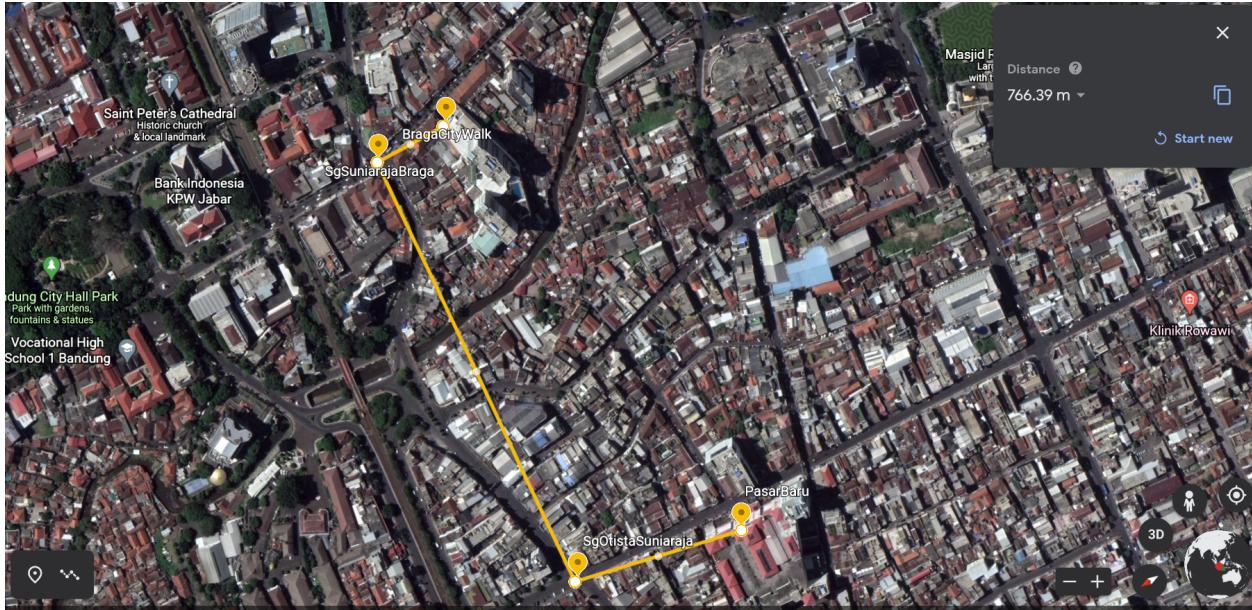
Setelah memasukan nama file external dengan format (namafile.txt), program akan mengeluarkan visualisasi graf peta file external. Bentuk peta dapat tidak rapi dikarenakan posisi node disesuaikan dengan koordinat longitude dan latitude yang dikonversi ke koordinat kartesian yang memungkinkan jarak yang terlalu dekat menyebabkan node hampir berhimpit. Label angka di tiap sisi merupakan jarak garis lurus antar node.



Untuk melanjutkan program, *exit* visualisasi graf dan program akan menampilkan daftar node dan meminta masukan source dan destination node yang ingin kita cek. Akan di cek shortest path dari BragaCityWalk ke Pasar Baru. **Disclaimer: shortest path berdasarkan node yang sudah ditambahkan di program dan program tidak menghandle jika ada jalan yang satu arah atau dilarang melintas.**



Hasil keluaran program adalah shortest path dan visualisasinya, serta jarak yang harus ditempuh, yaitu BragaCityWalk → SgSuniarajaBraga (Simpang Jl.Suniaraja - Braga) → SgOtistaSuniaRaja(Simpang Otista-Suniaraja) → PasarBaru , dengan jarak 0.766 km. Berikut ini adalah perbandingan dengan jarak node sebenarnya dengan google earth.



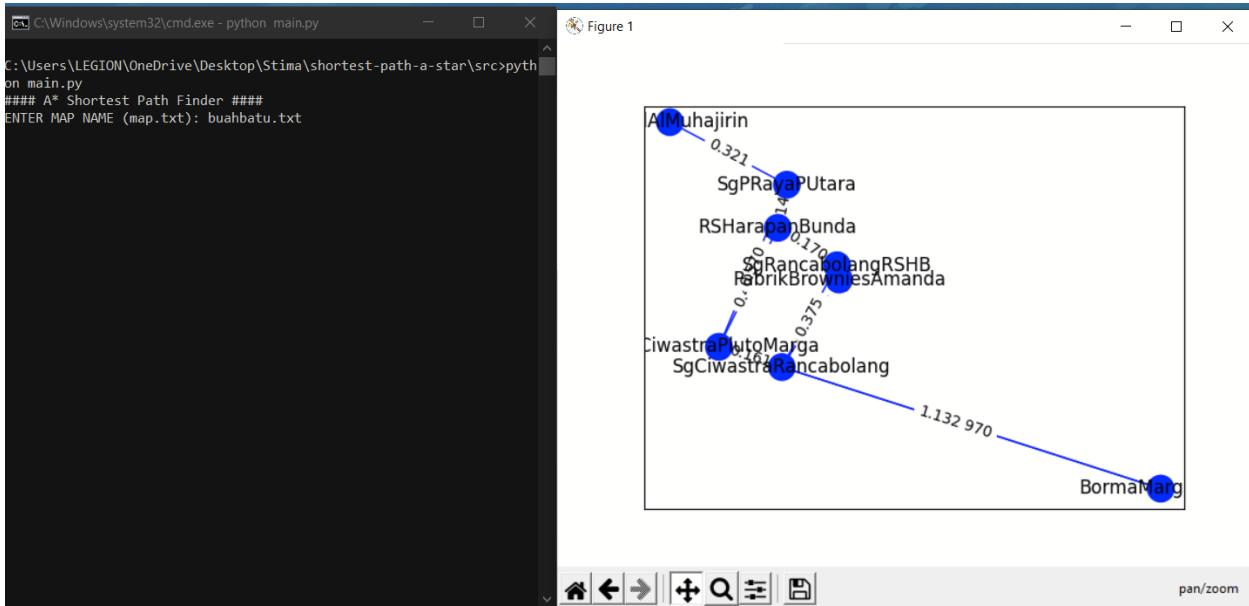
3. buahbatu.txt

```

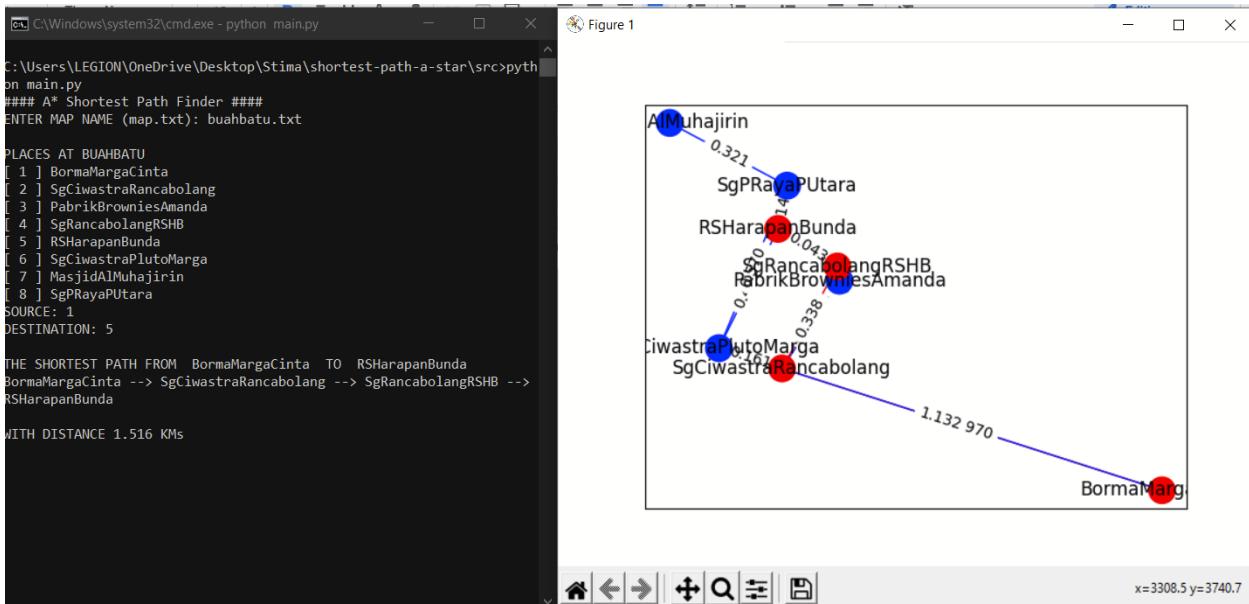
8
BormaMargaCinta -6.9551799892199035 107.6519276693611
SgCiwastraRancabolang -6.959027313829701 107.65981891843917
PabrikBrowniesAmanda -6.956031773373593 107.66033921920959
SgRancabolangRSHB -6.955743060729726 107.66060063498446
RSHarapanBunda -6.955911807122251 107.66213210141098
SgCiwastraPlutoMarga -6.959656497655654 107.66113547852977
MasjidAlMuhajirin -6.955354207774072 107.66552884470308
SgPRayaPUtara -6.95476406184648 107.66268484090399
0 1 0 0 1 0 0
1 0 1 1 0 1 0 0
0 1 0 1 0 0 0 0
0 1 1 0 1 0 0 0
0 0 0 1 0 1 0 1
1 1 0 0 1 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 1 1 1 0

```

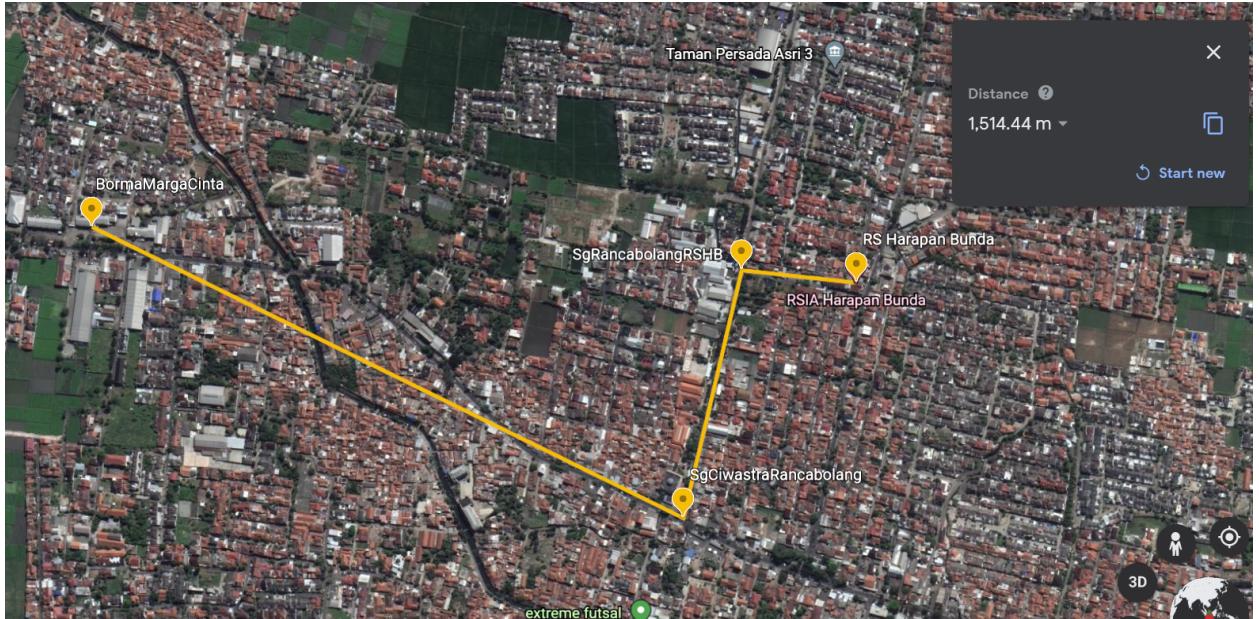
Setelah memasukan nama file external dengan format (namafile.txt), program akan mengeluarkan visualisasi graf peta file external. Bentuk peta dapat tidak rapi dikarenakan posisi node disesuaikan dengan koordinat longitude dan latitude yang dikonversi ke koordinat kartesian yang memungkinkan jarak yang terlalu dekat menyebabkan node hampir berhimpit. Label angka di tiap sisi merupakan jarak lurus antar node.



Untuk melanjutkan program, *exit* visualisasi graf dan program akan menampilkan daftar node dan meminta masukan source dan destination node yang ingin kita cek. Akan di cek shortest path dari Borma MargaCinta ke RS Harapan Bunda. **Disclaimer: shortest path berdasarkan node yang sudah ditambahkan di program dan program tidak menghandle jika ada jalan yang satu arah atau dilarang melintas.**



Hasil keluaran program adalah shortest path dan visualisasinya, serta jarak yang harus ditempuh, yaitu BormaMargaCinta → SgCiwastraRancabolang (Simpang Jl.Ciwastra - Rancabolang) → SgRancabolangRSHB(Simpang Rancabolang - jalan kecil penghubung Rancabolang - RS Harapan Bunda) → RSHarapanBunda, dengan jarak 1.516 km. Berikut ini adalah perbandingan dengan jarak node sebenarnya dengan google earth.



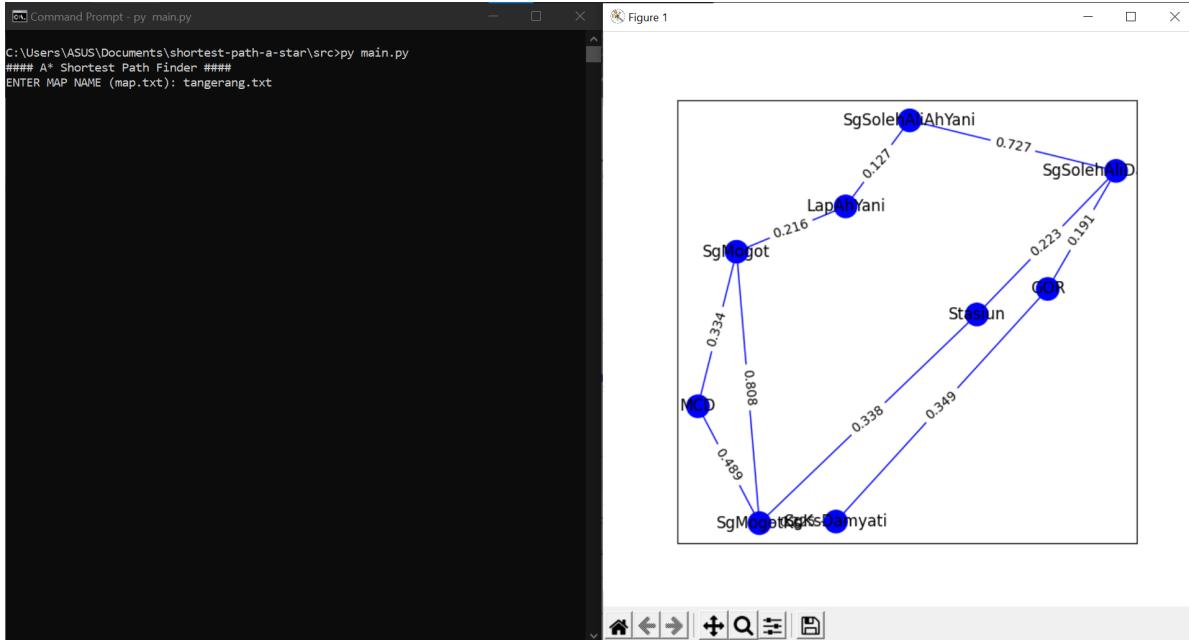
4. tangerang.txt

```

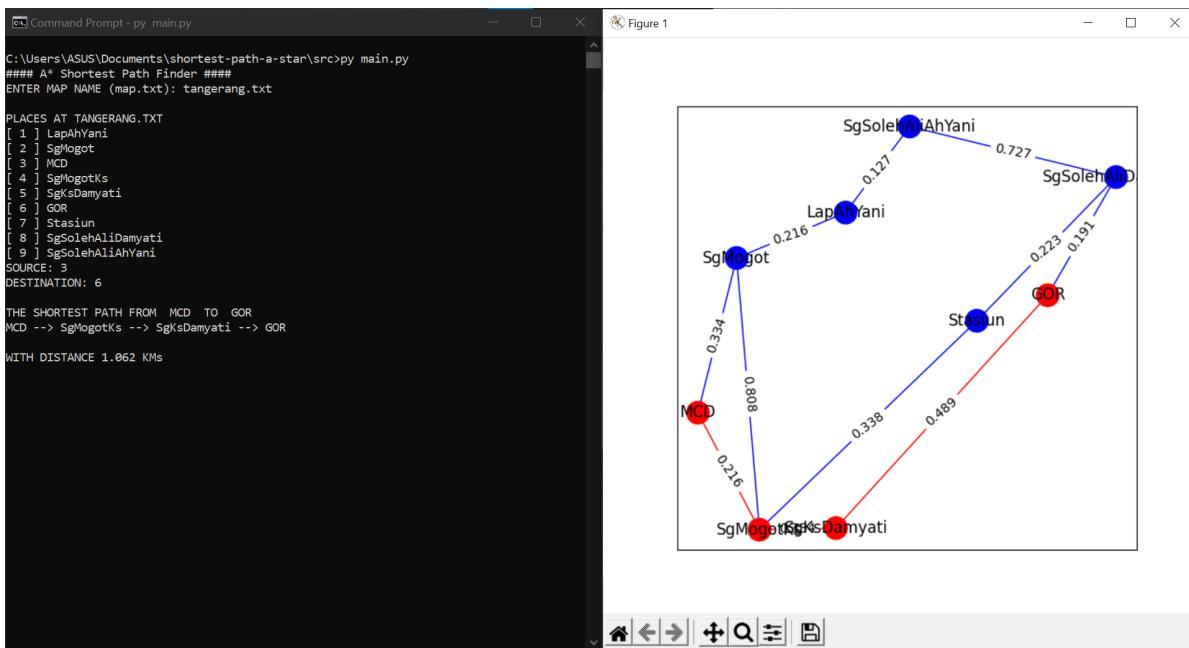
9
LapAhYani -6.171034876040317 106.63405218570935
SgMogot -6.169202277962933 106.63341102790366
MCD -6.171444092005021 106.63140107694329
SgMogotKs -6.175586907344772 106.62993055181754
SgKsDamyati -6.177613423067104 106.6299920128765
GOR -6.178159025211438 106.63309637679686
Stasiun -6.176801597885472 106.63273056128227
SgSolehAliDamyati -6.177405618504051 106.6346512135528
SgSolehAliAhYani -6.170889811884585 106.63519363668155
0 1 0 0 0 0 0 1
1 0 1 1 0 0 0 0
0 1 0 1 0 0 0 0
0 1 1 0 1 0 1 0
0 0 0 1 0 1 1 0
0 0 0 0 1 0 0 1 0
0 0 0 1 1 0 0 1 0
0 0 0 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0

```

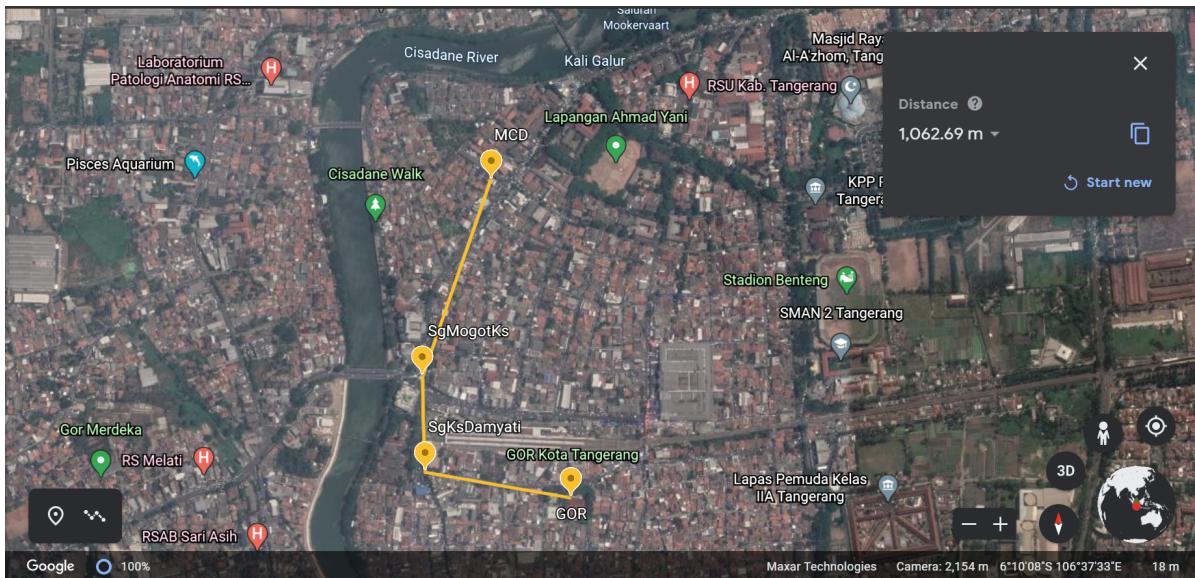
Setelah memasukan nama file external dengan format (namafile.txt), program akan mengeluarkan visualisasi graf peta file external. Bentuk peta dapat tidak rapi dikarenakan posisi node disesuaikan dengan koordinat longitude dan latitude yang dikonversi ke koordinat kartesian yang memungkinkan jarak yang terlalu dekat menyebabkan node hampir berhimpit. Label angka di tiap sisi merupakan jarak garis lurus antar node.



Untuk melanjutkan program, *exit* visualisasi graf dan program akan menampilkan daftar node dan meminta masukan source dan destination node yang ingin kita cek. Akan di cek shortest path dari MCD Daan Mogot ke GOR Tangerang. **Disclaimer: shortest path berdasarkan node yang sudah ditambahkan di program dan program tidak menghandle jika ada jalan yang satu arah atau dilarang melintas.**



Hasil keluaran program adalah shortest path dan visualisasinya, serta jarak yang harus ditempuh, yaitu MCD → SgMogotKs (Simpang Mogot-Kisamaun) → SgKsDamyati (Simpang Kisamaun-A.Damyati) → GOR (Jl. A.Damyati), dengan jarak 1.062 km. Berikut ini adalah perbandingan dengan jarak node sebenarnya dengan google earth.



TAUTAN SOURCE CODE

Github : <https://github.com/alifbhadrka/shortest-path-a-star>

TABEL PENILAIAN

Poin	Ya
1. Program dapat menerima input graf	✓
2. Program dapat menghitung lintasan terpendek	✓
3. Program dapat menampilkan lintasan terpendek serta jaraknya	✓
4. Bonus: <i>Program dapat menerima input peta dengan Google Map API dan menampilkan peta</i>	