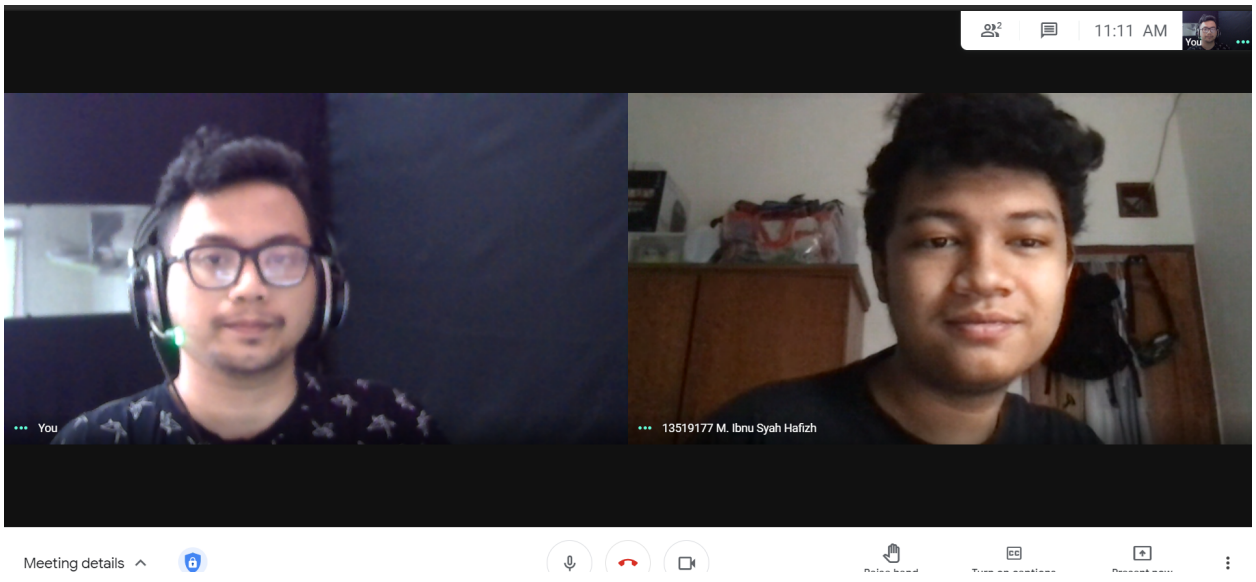
LAPORAN TUGAS KECIL
IF2211 STRATEGI ALGORITMA
IMPLEMENTASI ALGORITMA A* UNTUK MENENTUKAN LINTASAN TERPENDEK



Disusun oleh:
13519177 M. Ibnu Syah Hafizh
13519196 Rayhan Asadel

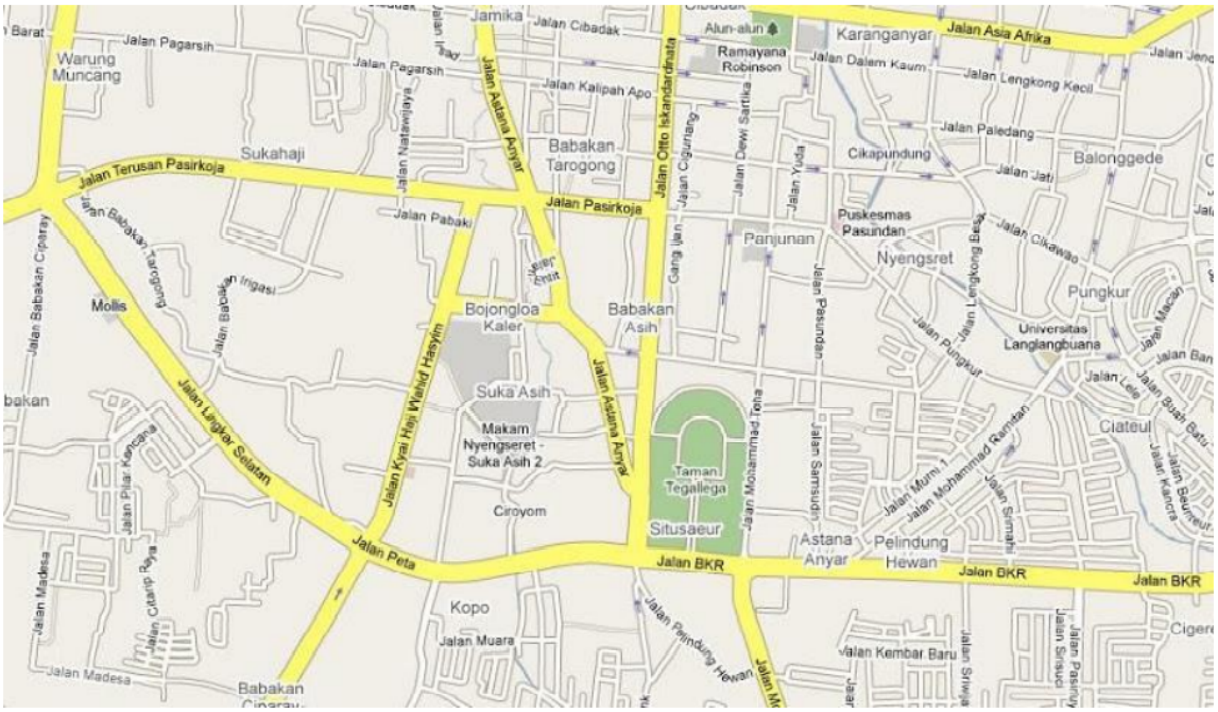
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

Daftar Isi

BAB I

DESKRIPSI TUGAS

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 berdasarakan 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.



Gambar 1.1 : Contoh Peta di Google Map

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar kampus ITB). Sisi diperoleh dari jalan antar dua simpul dan bobot sisi adalah jarak Euclidean. Berdasarkan graf yang dibentuk, lalu program A*

menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya. Lintasan terpendek dapat ditampilkan pada peta/graf. Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan. Spesifikasi program: 1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah. 2. Program dapat menampilkan peta/graf 3. Program menerima input simpul asal dan simpul tujuan. 4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan

BAB II

KODE PROGRAM

```
#Main.py
from os import name
import math
import matplotlib.pyplot as plt
import networkx as nx


#Parsing file menjadi array of node, format : [Name, X, Y, {Neighbors : Bobot}]
def inputFile():
   print()
   filename = str(input("Masukan Nama File (Nama.txt): "))
   arrFile = []

   #Loop masukin baris ke array arrFile
   with open('../test/'+filename,'r') as file:
       for line in file:
           arrFile.append(line.replace('\n', '').replace('(','').replace(')',''))

   X = 0
   Y = 0
   NamaSimpul = str("")
   ArrSimpul = []      #Arr berisi list (Nama, X, Y, {tetangga : bobot}) dari simpul
   jmlNode = int(arrFile[0])
   MatriksAdjacency = [ [ 0 for i in range(jmlNode) ] for j in range(jmlNode) ]

   #loop mengisi ArrSimpul berisi list (Nama, X, Y) dari simpul
   for i in range (1, jmlNode+1):
       #print(arrFile[i])
       stringTempCoor = str("")
       for j in range (len(arrFile[i])):

           if (arrFile[i][j] == ','):
               X = float(stringTempCoor)
               stringTempCoor = ""

           elif (arrFile[i][j] == ' '):
```

```
                Y = float(stringTempCoor)
                stringTempCoor = ""
            elif (arrFile[i][j] != ',' and arrFile[i][j] != ' '):
                stringTempCoor = stringTempCoor+arrFile[i][j]
                NamaSimpul = stringTempCoor
        ArrSimpul.append([NamaSimpul,X,Y])

    NilaiBobot = float(0)
    k = int(0)
    l = int(0)

    #Mengisi matriks adjacency dengan bobot sesuai inputfile
    for i in range (jmlNode+1, len(arrFile)):
        #print(arrFile[i])
        stringTempCoor = str("")
        l = 0
        for j in range (len(arrFile[i])):
            if (arrFile[i][j] == '[' and arrFile[i][j] == "]"):
                continue
            elif (arrFile[i][j] == ' ' or arrFile[i][j] ==']'):
                NilaiBobot = float(stringTempCoor)
                MatriksAdjacency[k][l] = NilaiBobot
                stringTempCoor = ""
                l = l+1
            elif (arrFile[i][j] != '[' and arrFile[i][j] != ' ' and arrFile[i][j] !=
']'):
                stringTempCoor = stringTempCoor+arrFile[i][j]
        k = k+1
    # print("Matriks Adjacency")
    # print(MatriksAdjacency)

    #Memasukan dictionary ketetanggan pada arr simpul, menjadi list (Nama, X, Y,
{Simpul : bobot}) dari simpul
    arrNodeName = []
    for i in range (len(ArrSimpul)):
        arrNodeName.append(ArrSimpul[i][0])
    for i in range(len(ArrSimpul)):
        temp = {}
        k = 0
        for j in range (len(MatriksAdjacency[i])):
            temp[arrNodeName[k]] = MatriksAdjacency[i][j]
            k+=1
```

```
        neightbors = {}
        for l in temp.keys():
            if (temp[l] != 0):
                neightbors[l] = temp[l]
        (ArrSimpul[i]).append(neightbors)

    return (ArrSimpul)

class Node:
    def __init__(self, name:str, parent:str, x:float, y:float, neightbor:dict):
        self.name = name
        self.parent = parent
        self.neightbor = neightbor
        self.x = x # koordinat X
        self.y = y # koordinat Y
        self.g = 0 # jarak dari simpul asal
        self.h = 0 # jarak ke simpul tujuan
        self.f = 0 # Total jarak
    # Compare
    def __eq__(self, other:str):
        return self.name == other
    # Sort
    def __lt__(self, other):
         return self.f < other.f
    # Print
    def __repr__(self):
        return (self.name)

def Haversine(node1:Node, node2:Node):
    lon1 =  math.radians(node1.x)
    lon2 =  math.radians(node2.x)
    lat1 =  math.radians(node1.y)
    lat2 =  math.radians(node2.y)
    R = 6373000
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c
    return distance

class Graph:
```

```
    def __init__(self, graph_dict=None, directed=True):
        self.graph_dict = graph_dict or {}
        self.directed = directed
        if not directed:
            self.make_undirected()
    def make_undirected(self):
        for a in list(self.graph_dict.keys()):
            for (b, dist) in self.graph_dict[a].items():
                self.graph_dict.setdefault(b, {})[a] = dist
    def connect(self, A, B, distance = 1):
        self.graph_dict.setdefault(A, {})[B] = distance
        if not self.directed:
            self.graph_dict.setdefault(B, {})[A] = distance
    # Get neighbors or a neighbor
    def get(self, a, b=None):
        neighbor = self.graph_dict.setdefault(a, {})
        if b is None:
            return neighbor
        else:
            return neighbor.get(b)

def make_ArrayofNode():
    file = inputFile()
    arrNode = []
    for i in range (len(file)):
        arrNode.append(Node(file[i][0], None, file[i][1], file[i][2], file[i][3]))
    return arrNode
 # A* search
def Astar(graph, heuristics, start, end, arrayOfNode):
    openList = [] # Simpul hidup
    closedList = [] # Simpul yang sudah dikunjungi
    # Inisiasi simpul start dan goal
    start_node = Node(start, None, None, None, None)
    goal_node = Node(end, None, None, None, None)
    # Masukkan simpul start ke openList
    openList.append(start_node)

    # Loop hingga openList kosong
    while len(openList) > 0:
        openList.sort()
        current_node = openList.pop(0) # Node dengan jarak terkecil
        closedList.append(current_node) # Masukkan ke closedList
```

```
        if current_node == goal_node: # Jika node yang di cek adalah simpul tujuan
            path = []
            while current_node != start_node:
                path.append(current_node.name + ': ' + str(current_node.g))
                current_node = current_node.parent
            path.append(start_node.name + ': ' + str(start_node.g))
            return path[::-1] #return reversed path
        # Get neighbours
        neighbors = graph.get(current_node.name)
        # Loop neighbors
        for key,value in neighbors.items():
            for i in range (len(arrayOfNode)):
                if (arrayOfNode[i].name == key):
            # Inisiasi node neighbor dengan parent current_node
                    neighbor = Node(key, current_node, arrayOfNode[i].x,
arrayOfNode[i].y, arrayOfNode[i].neightbor)
            # Check apakah neighbor di dalam closedList
            if(neighbor in closedList):
                continue # Abaikan
            # Hitung f(n)
            neighbor.g = current_node.g + graph.get(current_node.name, neighbor.name)
            neighbor.h = heuristics.get(neighbor.name)
            neighbor.f = neighbor.g + neighbor.h
            # Check apakah neighbor harus dimasukkan ke openList
            if(CheckAddNode(openList, neighbor) == True):
                openList.append(neighbor)
    return None # Tidak terdapat lintasan

def CheckAddNode(open, neighbor):
    for node in open:
        if (neighbor == node and neighbor.f > node.f):
            return False
    return True

def main():
    arrNode = make_ArrayofNode()
    graph = Graph()
     haversine = {}
    for i in range (len(arrNode)):
        haversine[arrNode[i].name] = Haversine(arrNode[0], arrNode[i])

    for i in range (len(arrNode)):
```

```
    for j in arrNode[i].neightbor:
        graph.connect(arrNode[i].name, j, arrNode[i].neightbor[j])

print("Daftar tempat/persimpangan jalan : ")
for i in range (len(arrNode)):
    print(str(i+1)+ '.', arrNode[i])
print()
print("Masukkan Lokasi!")
inputNode1 = str(input("Lokasi Awal : "))
inputNode2 = str(input("Lokasi Tujuan : "))

path = Astar(graph, haversine, inputNode1, inputNode2, arrNode)
print()
print("Lintasan :", path)
# print(haversine)


arraynamanode = []
temp =""
j
for i in range (len(path)):
    j =0
    temp =""
    while (path[i][j]!=":"):
        temp = temp+ str(path[i][j])
        j=j+1

    arraynamanode.append(temp)

arrayBobot = []
tempbobot =""
for i in range (len(path)):
    j = 0
    tempbobot =""
    while (path[i][j]!=' '):
        j=j+1
    j += 1
    while (j != len(path[i])):
        tempbobot = tempbobot+ str(path[i][j])
        j=j+1
    arrayBobot.append(float(tempbobot))
```

```
   print("Total jarak terpendek antara", inputNode1, "dan", inputNode2, "adalah",
arrayBobot[len(arrayBobot)-1], "meter")

   G = nx.Graph()
   for i in range(len(path)):
       for j in range (len(arrNode)):
           if (arraynamanode[i]==arrNode[j].name):
               G.add_node(i,pos=(float(arrNode[j].x),float(arrNode[j].y)))
   i = 0
   j = 1
   while (j<len(path)):
       G.add_edge(i,j,weight=arrayBobot[j]-arrayBobot[i])
       i = j
       j = j+1

   raw_labels = arraynamanode
   lab_node = dict(zip(G.nodes, raw_labels))
   pos=nx.get_node_attributes(G,'pos')
   #nx.draw(G,pos)
   labels = nx.get_edge_attributes(G, "weight")
   nx.draw_networkx_edges(G, pos, edge_color='g',arrows=False)
   nx.draw_networkx_nodes(G, pos, node_color='r',node_size=500)
   nx.draw_networkx_edge_labels(G, pos, edge_labels=labels,font_size=8)
   nx.draw_networkx_labels(G, pos, labels=lab_node, font_size=10, font_family="DejaVu
Sans")
   plt.show()


if __name__ == "__main__": main()
```

BAB III

HASIL PENGUJIAN

Kasus 1 : Dago/sekitar ITB

Nama file : itb.txt

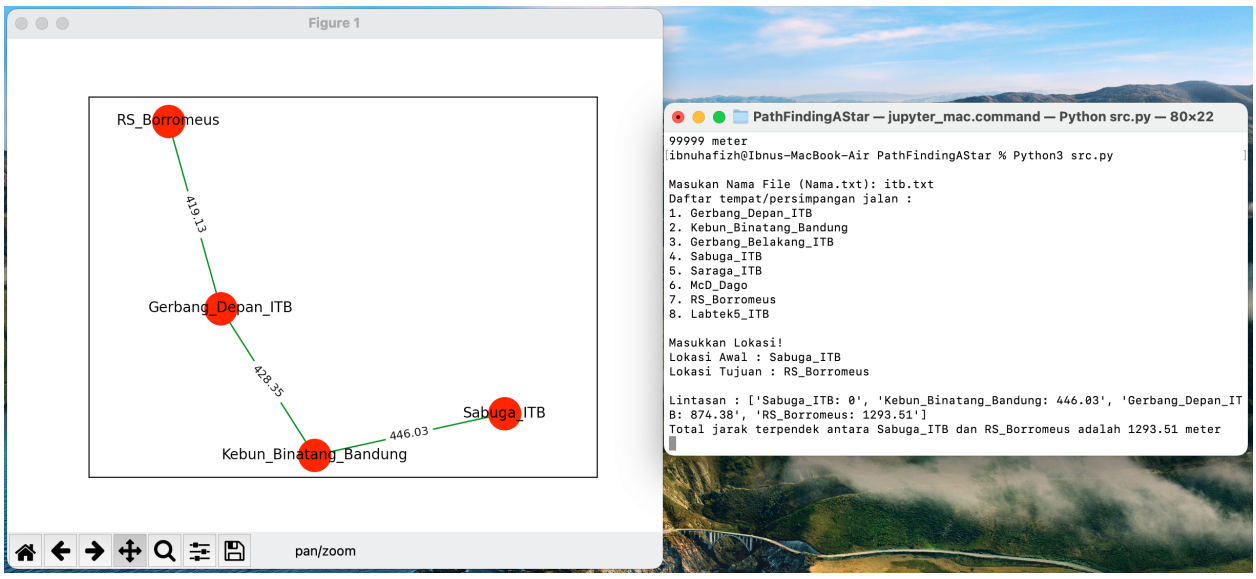
Isi file :

itb.txt

```
8
(-6.892616,107.610423) Gerbang_Depan_ITB
(-6.890464174304161,107.60724717865062) Kebun_Binatang_Bandung
(-6.887407,107.611507) Gerbang_Belakang_ITB
(-6.886107754151466,107.60814840083982) Sabuga_ITB
(-6.886011891139484,107.61040145641657) Saraga_ITB
(-6.884371565786224,107.61305147865777) McD_Dago
(-6.893819336137485,107.61446768497895) RS_Borromeus
(-6.890591990423812,107.60992938762139) Labtek5_ITB
[0 428.35 0 0 0 0 419.13 233.94]
[428.35 0 584.79 446.03 0 0 0 0]
[0 584.79 0 427.57 193.57 0 0 398.50]
[0 446.03 427.57 0 310.81 0 0 0]
[0 0 193.57 310.81 0 339.15 0 0]
[0 0 0 0 339.15 0 1000.00 0]
[419.13 0 0 0 0 1000.00 0 0]
[233.94 0 398.50 0 0 0 0 0]
```

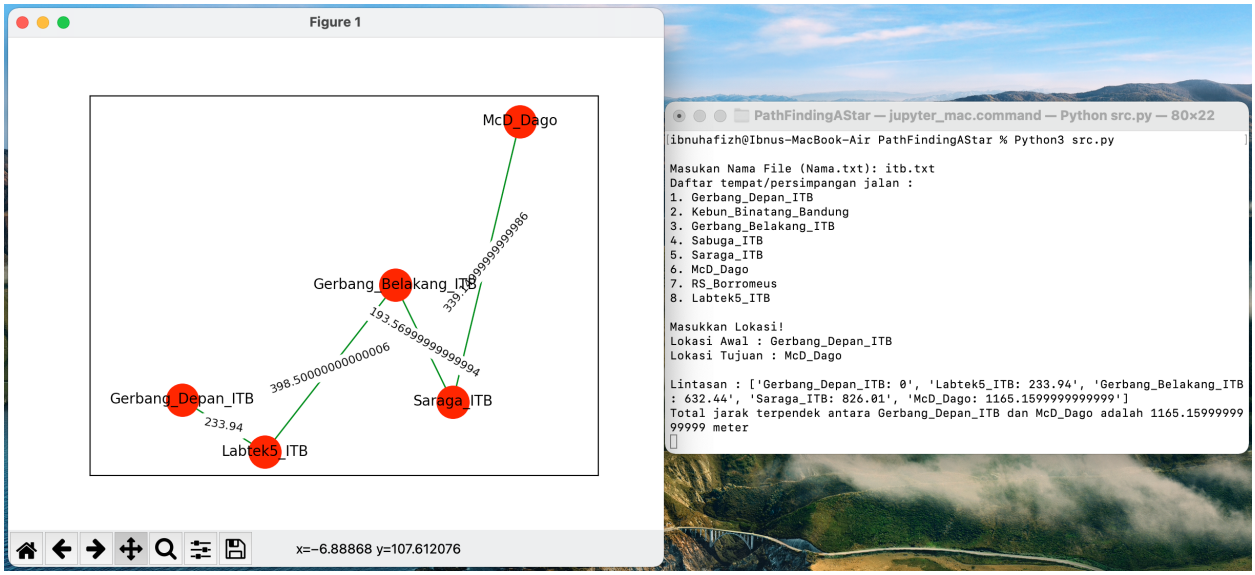
Output 1 :

Shortest path dari Sabuga_ITB ke RS_Borromeus



Output 2:

Shortest path dari Gerbang_Depan_ITB ke Mcd_Dago



Kasus 2 : Area Alun-Alun Bandung

Nama file : bandung.txt

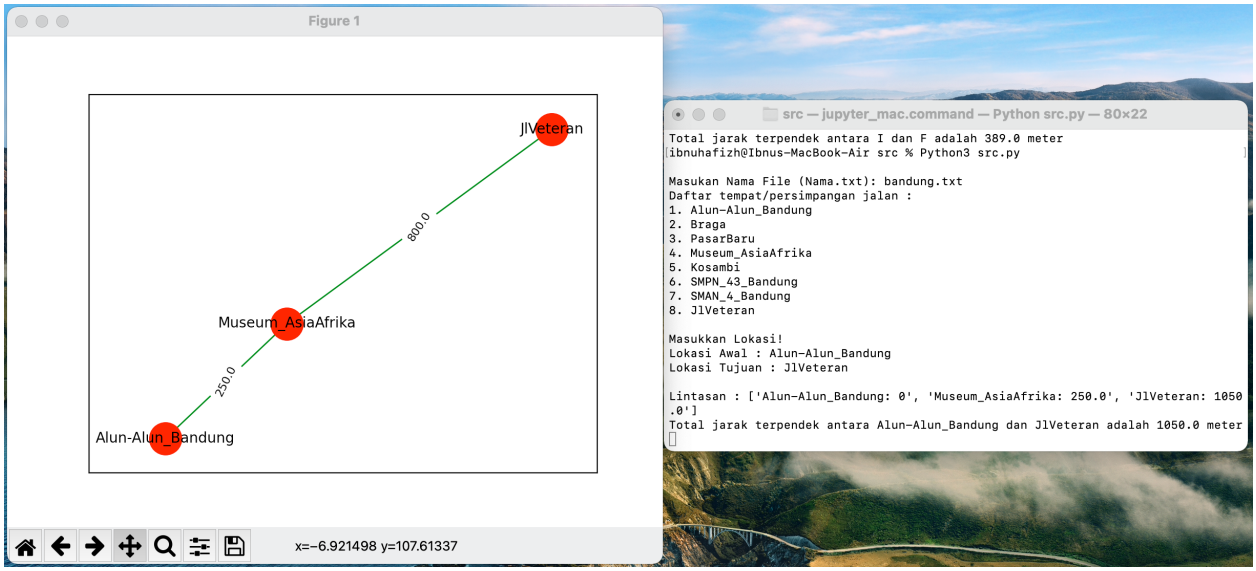
Isi file :

test > bandung.txt

```
8
(-6.921957744344849,107.60703707893073) Alun-Alun_Bandung
(-6.915063849950349,107.6089459921182) Braga
(-6.917555347958105,107.60353104451164) PasarBaru
(-6.9209579997254025,107.60968467023457) Museum_AsiaAfrika
(-6.924036914857504,107.61413208913999) Kosambi
(-6.925451794947383,107.60687488707937) SMPN_43_Bandung
(-6.919332245042484,107.59885190047153) SMAN_4_Bandung
(-6.918774961513411,107.61416226185753) JlVeteran
[0 1000 700 250 0 550 1000 0]
1000 0 750 450 0 0 0 850]
[700 750 0 0 800 1200 900 0]
[250 450 0 0 0 0 0 800]
[0 0 0 800 0 1100 0 900]
[550 0 1200 0 1100 0 1600 0]
[1000 0 900 0 0 1600 0 0]
[0 850 0 800 900 0 0 0]
```

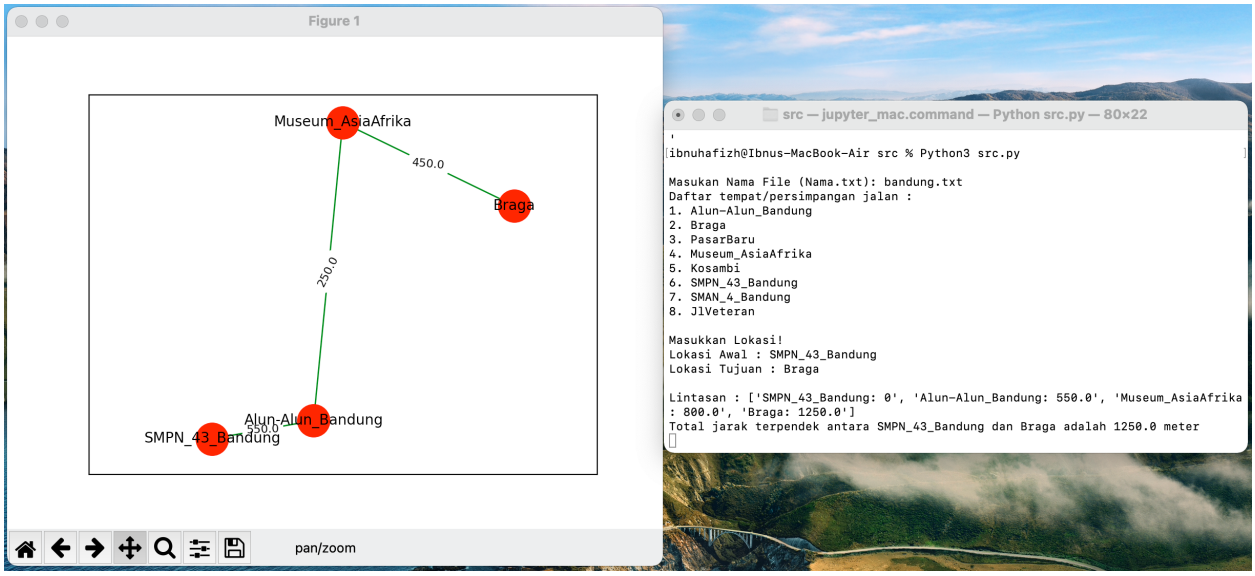
Output 1 :

Shortest path dari Alun-Alun_Bandung ke JlVeteran



Output 2:

Shortest path dari SMPN_43_Bandung ke Braga



Kasus 3 : Daerah Buah Batu Bandung

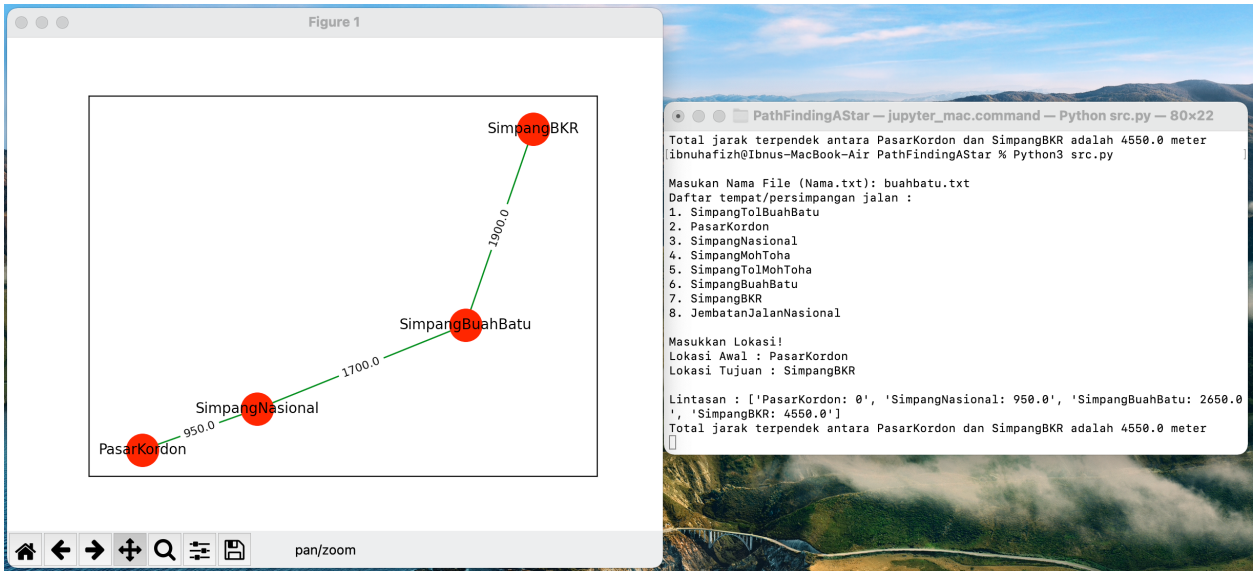
Nama file : buahbatu.txt

Isi file :

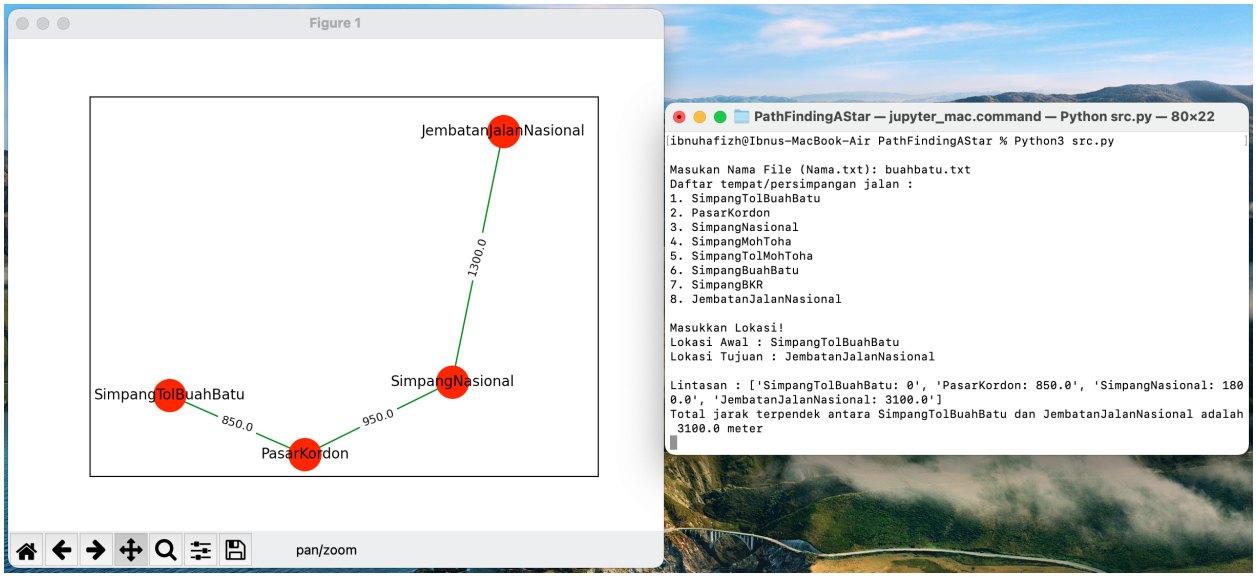
buahbatu.txt

```
8
(10442777,-3320391) SimpangTolBuahBatu
(10444121,-3320913) PasarKordon
(10445584,-3320274) SimpangNasional
(10446931,-3315857) SimpangMohToha
(10445445,-3315865) SimpangTolMohToha
(10448238,-3318957) SimpangBuahBatu
(10449102,-3315890) SimpangBKR
(10446092,-3318073) JembatanJalanNasional
[0 850 0 0 4700 0 0 0]
[850 0 950 0 0 0 0 0]
[0 950 0 0 0 1700 0 1300]
[0 0 0 0 854 0 1300 2300]
[4700 0 0 854 0 0 0 0]
[0 0 1700 0 0 0 1900 0]
[0 0 0 1300 0 1900 0 0]
[0 0 1300 2300 0 0 0 0]
```

Output 1 :
Shortest path dari PasarKordon ke SimpangBKR



Output 2 :
Shortest path dari SimpangTolBuahBatu ke JembatanJalanNasional



Kasus 4 : Kota Bogor

Nama file : bogor.txt

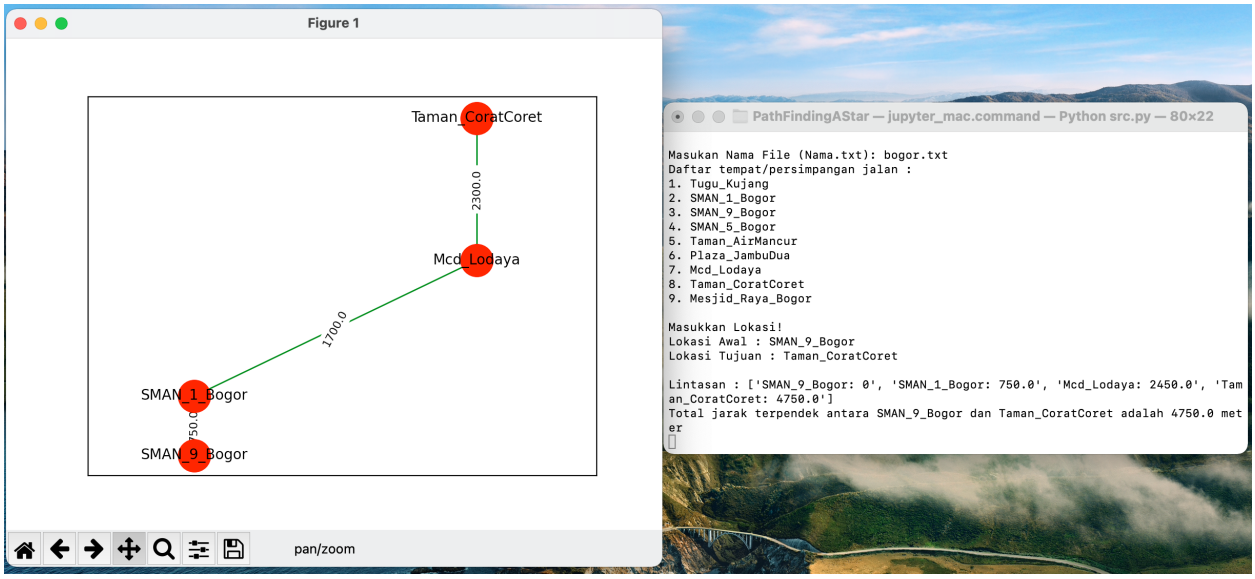
Isi file :

bogor.txt

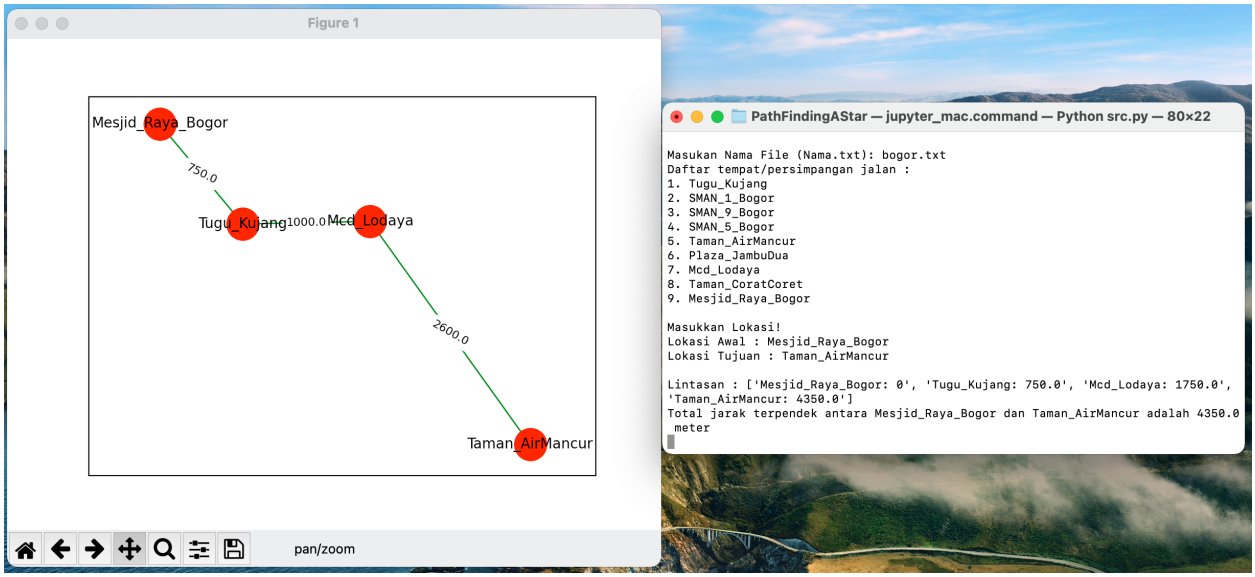
```
9
(-6.601190205706389,106.80505910220238) Tugu_Kujang
(-6.597102267767196,106.79326552175363) SMAN_1_Bogor
(-6.5961016007735225 106.78808654683198) SMAN_9_Bogor
(-6.585340741224928,106.78374527464165) SMAN_5_Bogor
(-6.581182320850733,106.79660038973282) Taman_AirMancur
(-6.569052566422009,106.8080741858986) Plaza_JambuDua
(-6.592326783361264,106.80515752839156) Mcd_Lodaya
(-6.5818813524410995 106.81756945868038) Taman_CoratCoret
(-6.606984542281702,106.80889402039062) Mesjid_Raya_Bogor
[0 1900 0 0 0 0 1000 0 750]
[1900 0 750 0 2000 0 1700 0 0]
[0 750 0 1400 0 0 0 0 0]
[0 0 1400 0 1700 0 0 0 0]
[0 2000 0 1700 0 2100 2600 0 0]
[0 0 0 0 2100 0 0 2300 0]
[1000 1700 0 0 2600 2900 0 2300 0]
[0 0 0 0 0 2300 2300 0 4200]
[750 0 0 0 0 0 0 4200 0]
```

Output 1 :

Shortest path dari SMAN_9_Bogor ke Taman_CoratCoret



Output 2 :
Shortest path dari Mesjid_Raya_Bogor ke Taman_AirMancur



Kasus 5 : Random Case

Nama file : case1.txt

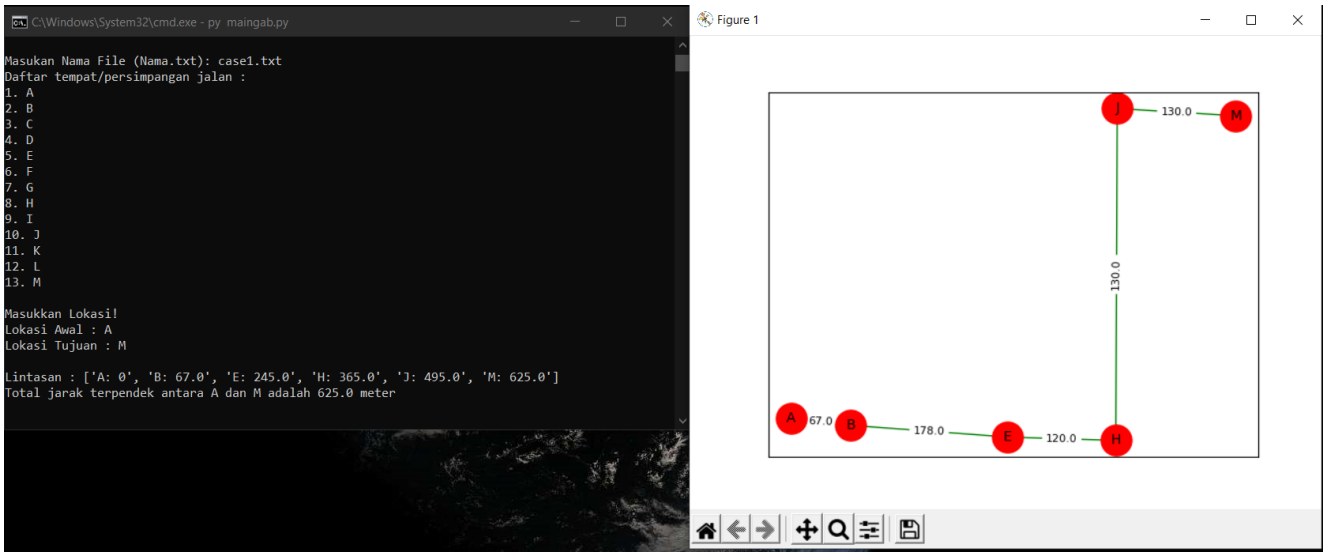
Isi file :

case1.txt

```
13
(-6.893220,107.610454) A
(-6.892613,107.610428) B
(-6.892625,107.608842) C
(-6.891049,107.608705) D
(-6.891012,107.610386) E
(-6.890995,107.611570) F
(-6.892568,107.611680) G
(-6.889913,107.610374) H
(-6.889943,107.609006) I
(-6.889898,107.611563) J
(-6.888721,107.609086) K
(-6.888697,107.610370) L
(-6.888690,107.611537) M
[0 67 0 0 0 0 0 0 0 0 0 0 0]
[67 0 174 0 178 0 140 0 0 0 0 0 0]
[0 174 0 175 0 0 0 0 0 0 0 0 0]
[0 0 175 0 192 0 0 0 151 0 0 0 0]
[0 178 0 192 0 142 0 120 0 0 0 0 0]
[0 0 0 0 142 0 175 0 0 121 0 0 0]
[0 140 0 0 0 175 0 0 0 0 0 0 0]
[0 0 0 0 120 0 0 0 149 130 0 135 0]
[0 0 0 151 0 0 0 149 0 0 139 0 0]
[0 0 0 0 0 121 0 130 0 0 0 0 130]
[0 0 0 0 0 0 0 0 139 0 0 143 0]
[0 0 0 0 0 0 0 135 0 0 143 0 129]
[0 0 0 0 0 0 0 0 130 0 129 0]
```

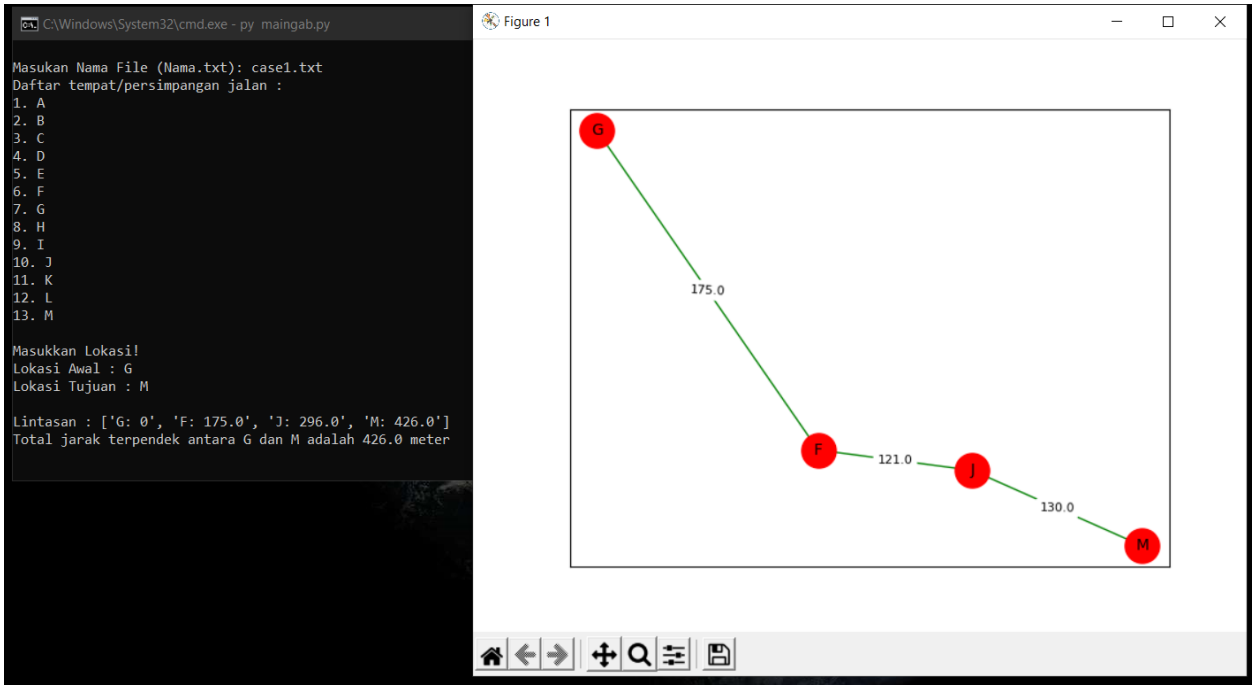
Output 1 :

Path dari titik A ke titik M



Output 2 :

Path dari titik G ke titik M



Kasus 6 : Random Case

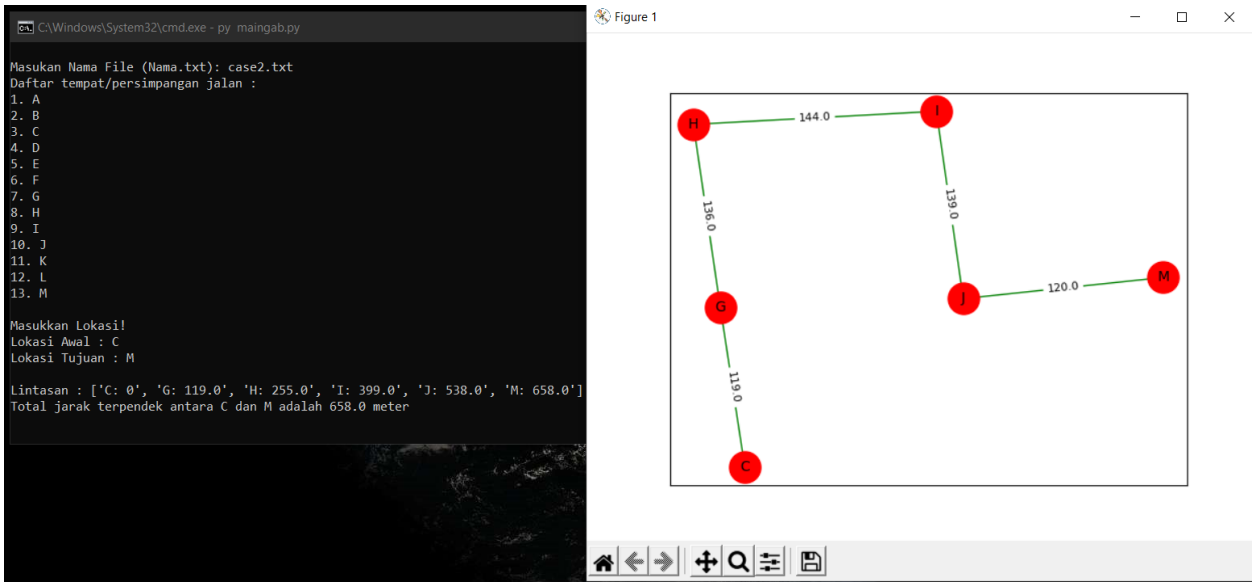
Nama file : case2.txt

Isi file :

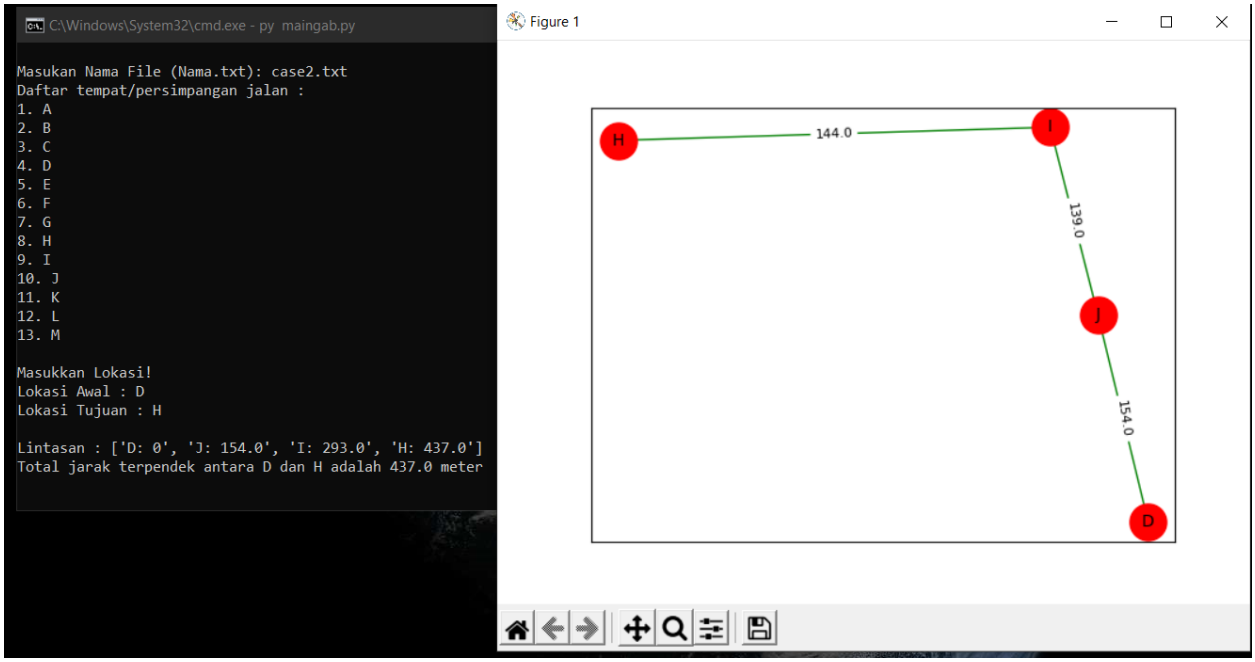
case2.txt

```
13
(-6.922019,107.604031) A
(-6.920773,107.604115) B
(-6.922231,107.605329) C
(-6.920906,107.605070) D
(-6.923253,107.605288) E
(-6.923342,107.606328) F
(-6.922362,107.606395) G
(-6.922507,107.607622) H
(-6.921202,107.607710) I
(-6.921058,107.606456) J
(-6.921224,107.608059) K
(-6.920178,107.608205) L
(-6.919983,107.606598) M
[0 138 145 0 0 0 0 0 0 0 0 0 0]
[138 0 0 107 0 0 0 0 0 0 0 0 0]
[145 0 0 0 113 0 119 0 0 0 0 0 0]
[0 107 0 0 0 0 0 0 0 154 0 0 0]
[0 0 113 0 0 116 0 0 0 0 0 0 0]
[0 0 0 0 116 0 109 0 0 0 0 0 0]
[0 0 119 0 0 109 0 136 0 0 0 0 0]
[0 0 0 0 0 0 136 0 144 0 0 0 0]
[0 0 0 0 0 0 0 144 0 139 38 0 0]
[0 0 0 154 0 0 0 0 139 0 0 0 120]
[0 0 0 0 0 0 0 0 38 0 0 118 0]
[0 0 0 0 0 0 0 0 0 0 118 0 179]
[0 0 0 0 0 0 0 0 0 120 0 179 0]
```

Output 1 :
Path dari titik C ke titik M



Output 2 :
Path dari titik D ke titik H



Link Github Source Code: https://github.com/RayhanAsadel/PathFindingAStar.git