Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

(Penerapan *Route/Path Planning Algoritma A**)

Oleh :

Muhammad Fahkry Malta 13519032 K01

Muhammad Fikri. N 13519069 K02

INSTITUT TEKNOLOGI BANDUNG

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

TEKNIK INFORMATIKA

2020/2021

BAB I

DESKRIPSI MASALAH

Menentukan lintasan terpendek adalah proses pencarian sebuah lintasan yang memiliki jarak paling pendek dalam perjalanannya dari suatu tempat awal ke tempat tujuan. Permasalahan menentukan lintasan terpendek ini dapat dipecahkan dengan memanfaatkan beberapa algoritma, salah satunya adalah Algoritma A*. Algoritma A* memiliki ide untuk menghindari rute/jalan yang memang sudah memiliki jarak yang besar.

Contoh :

1. - Input file teks pertama berisi :

   8

   A 5,2

   B 4,1

   C 7,9

   D 1,5

   E 3,9

   F 6,7

   G 4,2

   H 7,6

   - Input file teks kedua berisi :

   0 1 1 0 0 0 0 0

   1 0 0 1 0 0 0 0

   1 0 0 0 0 1 0 0

   0 1 0 0 0 0 0 1

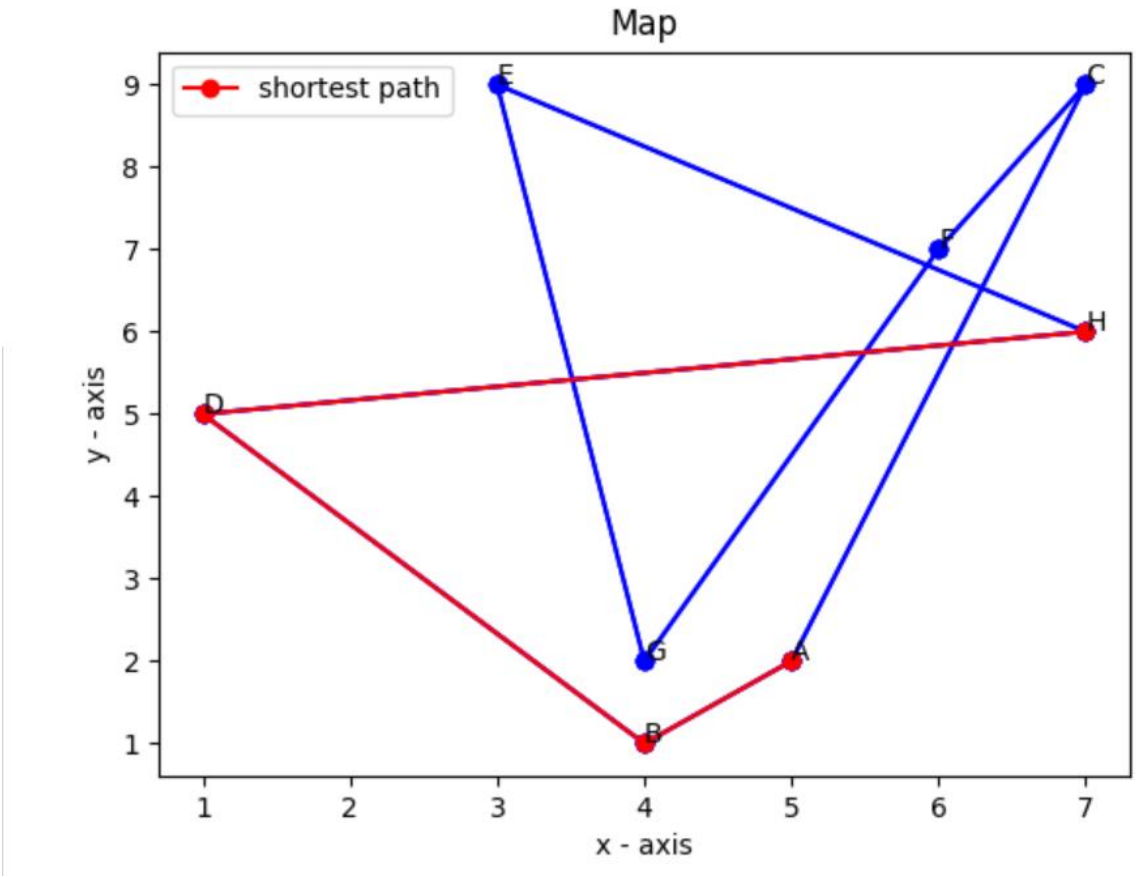
   0 0 0 0 0 0 1 1

   0 0 1 0 0 0 1 0

   0 0 0 0 1 1 0 0

0 0 0 1 1 0 0 0

Solusi/output-nya adalah :




```
----------------------------
|   Dari mana mau ke mana   |
|           ???             |
----------------------------
Input asal (Node Awal) : A
Input tujuan (Node Akhir) : H
Rute Terpendek :
rute :  A -> B -> D -> H
Iterasi : 8
Jarak : 12.49
```

Spesifikasi :

Membuat aplikasi sederhana yang dapat menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung, dengan memanfaatkan algoritma A* dalam bahasa C/C++/Java/Python/C#/Golang. 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. Aplikasi dapat menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah, dapat menampilkan peta/graf, menerima input simpul asal dan simpul tujuan, dan dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.

BAB II

ALGORITMA A*

Algoritma A* digunakan untuk menentukan lintasan terpendek dalam suatu graf/peta. Algoritma A* memiliki ide untuk menghindari rute/jalan yang memang sudah memiliki jarak yang besar

Bahasa pemrograman yang digunakan adalah Python.

Pemecahan masalah menentukan lintasan terpendek menggunakan algoritma A* :

1. Baca file teks pertama dan kedua kemudian hasilnya disimpan ke array yang berisi koordinat tiap simpul dan array *boolean* yang menyatakan ketetanggan antar masing-masing simpul.

([[nama simpul_1, koordinat x_1, koordinat y_1], [nama simpul_2, koordinat x_2, koordinat y_2], dst])

2. Algoritma A star

```
def a_star(nodeAwal, nodeAkhir,iterasi):
    iterasi += 1
    rute.append(nodeAwal)
    if (is_have_edge(nodeAwal) == 1 and iterasi != 1):
        if (len(arr) == 0):
            print("Tidak bisa mencapai lokasi tujuan")
            rute.append("NO")
        else:
            for i in range(len(mat_mix[dict[nodeAwal]])):
                if mat_mix[dict[nodeAwal]][i] != 0:

                    nodeN = get_key(i)

                    gn = calculate_gn(rute,nodeN)
                    # print(gn)
                    hn = g.matrix[i][dict[nodeAkhir]]
                    # print(hn)
                    total = gn + hn
                    # print(total)
                    rute.append(get_key(i))
                    copy_rute = ut.copy_arr(rute)
                    arr_total = [total,copy_rute]
                    arr.append(arr_total)
                    del rute[len(rute)-1:len(rute)]

            if (len(arr) != 0):
                idx_min = ut.min_arr(arr)
                temp = arr[idx_min]
                del arr[idx_min:idx_min + 1]
                node = temp[1][len(temp[1])-1]
                if (node == nodeAkhir):
                    print("Rute Terpendek : ")
```

```
                    jarak = calculate_gn(rute,node)
                    rute.append(node)
                    print("Rute : ", end=" ")
                    # print(rute)
                    ut.display_array(rute)
                    rute.append("YES")
                    print("Iterasi : " + str(iterasi))
                    print("Jarak : " + str(jarak))

                else:
                    rute.clear() # Clear Rute

                    for i in range(len(temp[1])):
                        rute.append(temp[1][i]) # Isi Kembali Rute

                    del rute[len(rute)-1:len(rute)]

                    # recursive
                    a_star(get_key(dict[temp[1][len(temp[1])-
1]]),nodeAkhir,iterasi)
            else: # klw node awal ga punya tetangga
                print("Tidak ada rute yang dapat dilalui")
                rute.append("NO")
            # else:
            #     print("Tidak bisa mencapai lokasi tujuan Anda")
    else:
        for i in range(len(mat_mix[dict[nodeAwal]])):
            if mat_mix[dict[nodeAwal]][i] != 0:
                nodeN = get_key(i)

                gn = calculate_gn(rute, nodeN)
                # print(gn)
                hn = g.matrix[i][dict[nodeAkhir]]
                # print(hn)
                total = gn + hn
                # print(total)
                rute.append(get_key(i))
                copy_rute = ut.copy_arr(rute)
                arr_total = [total, copy_rute]
                arr.append(arr_total)
                del rute[len(rute) - 1:len(rute)]

        if (len(arr) != 0):
            idx_min = ut.min_arr(arr)
            temp = arr[idx_min]
            # print("ARR")
            # print(arr)
            # print("TEMP")
            # print(temp)
            del arr[idx_min:idx_min + 1]
            node = temp[1][len(temp[1]) - 1]
            # print(node)
            if (node == nodeAkhir):
                print("Rute Terpendek : ")
                jarak = calculate_gn(rute, node)
                rute.append(node)
                print("rute : ", end=" ")
```

```
                # print(rute)
                ut.display_array(rute)
                rute.append("YES")
                print("Iterasi : " + str(iterasi))
                print("Jarak : " + str(jarak))

            else:
                rute.clear()  # Clear Rute

                for i in range(len(temp[1])):
                    rute.append(temp[1][i])  # Isi Kembali Rute
                # print("RITEEE")
                # print(rute)
                del rute[len(rute) - 1:len(rute)]
                # print("AFTER DELETE")
                # print(rute)

                # recursive
                a_star(get_key(dict[temp[1][len(temp[1]) - 1]]), nodeAkhir,
iterasi)
        else: # klw node awal ga punya tetangga
            print("Tidak ada rute yang dapat dilalui")
            rute.append("NO")
```

```
def Get_Short_Path(nodeAwal,nodeAkhir):
    iterasi = 0
    if (nodeAwal == nodeAkhir):
        print("Anda Sudah di sana :) ")
    else:
        if (is_have_edge(nodeAkhir) > 0): # Jika node akhir dapat dicapai
            a_star(nodeAwal,nodeAkhir,iterasi)
        else:
            print("Lokasi Tujuan tidak bisa dicapai")
```

BAB III

SOURCE CODE

File Main.py

```
import os
import matplotlib.pyplot as plt
import graph as gr
import util as ut

print("------------------------------")
print("|  Silahkan input nama file |")
print("|        (eg : tc1)         |")
print("|   (Tanpa ekstensi file)   |")
print("------------------------------")

print()

filename1 = input("Input nama file koordinat (format : *_1.txt) (eg : tc1_1)
: ")
test = "../test/"
path1 = test + filename1 + ".txt"
isfile1 = os.path.isfile(path1)

# Melakukan pengecekan apakah filename terdapat pada folder test atau tidak
while (not isfile1):
    filename1 = input("Ulangi input nama file koordinat (format : *_1.txt)
(eg : tc1_1) : ")
    path1 = test + filename1 + ".txt"
    isfile1 = os.path.isfile(path1)

filename2 = input("Input nama file Matriks adjacency (format : *_2.txt) (eg :
tc1_2) : ")
path2 = test + filename2 + ".txt"
isfile2 = os.path.isfile(path2)

while (not isfile2):
    filename2 = input("Ulangi input nama file 2 Matriks adjacency (format :
*_2.txt) (eg : tc1_2) : ")
    path2 = test + filename2 + ".txt"
    isfile2 = os.path.isfile(path2)

file_data = open(path1, "r")
file_edge = open(path2, "r")

data = file_data.read().splitlines() # baca file teks (dengan readlines yang
sekaligus hapus \n)
edge = file_edge.read().splitlines()

n_node = int(data[0])
data.remove(data[0])
int_data = [[0 for j in range(2)] for i in range(len(data))]

arr_node = []
for i in range(len(data)):
```

```
    # print("NODE")
    # print(node)
    idx_titik_dua = data[i].index(':')
    node = data[i][:(idx_titik_dua)]
    # print("idx" +str(idx_titik_dua))
    arr_node.append(node)
    string = data[i][idx_titik_dua + 1:len(data[i])]
    x = string.split(",")
    # print(x[0][0])
    # if (x[0][0] == "-"):
    if (x[0][0] == "-"):
        int_data[i][0] = -(float(x[0][1:]))
    else:
        int_data[i][0] = float(x[0])
    # int_data[i][0] = int(x[0])

    if (x[1][0] == "-"):
        int_data[i][1] = -(float(x[1][1:]))
    else:
        int_data[i][1] = float(x[1])

dict = {}
idx = 0
for node in arr_node:
    dict[node] = idx
    idx+= 1

mat_edge = [[0 for j in range(n_node)] for i in range(n_node)]

baris = 0
kolom = 0
for i in range(len(edge)):
    for j in range(len(edge[i])):
        if j % 2 == 0:
            mat_edge[baris][kolom] = int(edge[i][j])
            kolom += 1
    baris += 1
    kolom = 0

g = gr.Graf(n_node)

print("-----------------------------")
print("|   Silahkan input opsi     |")
print("|   1. Koordinat Peta       |")
print("|   2. Koordinat Kartesian  |")
print("-----------------------------")

pil = int(input("Input opsi koordinat : "))

while (pil != 1 and pil != 2):
    pil = int(input("Ulangi input pilihan, pastikan pilihan 1 atau 2 : "))
if pil == 2:
    # Membuat matriks koordinat node
    for i in range(len(data)):
        for j in range(i+1,len(data)):
            m = int_data[i]
```

```
            n = int_data[j]

            x1 = m[0]
            y1 = m[1]
            x2 = n[0]
            y2 = n[1]
            jarak = ut.Euclidean(x1,x2,y1,y2)
            g.add_element(i,j,jarak)
            g.add_element(j,i,jarak)
            g.add_element(i,i,0)
else:
    # Membuat matriks koordinat node
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            m = int_data[i]
            n = int_data[j]

            x_lat1 = m[0]
            y_lon1 = m[1]
            x_lat2 = n[0]
            y_lon2 = n[1]
            jarak = ut.Haversine(x_lat1, x_lat2, y_lon2, y_lon2)
            g.add_element(i, j, jarak)
            g.add_element(j, i, jarak)
            g.add_element(i, i, 0)

# print("MATRKS")
# ut.display_mat(g.matrix,n_node,n_node)

mat_mix = [[0 for j in range(n_node)] for i in range(n_node)]


for i in range(n_node):
    for j in range(n_node):
        if mat_edge[i][j] == 1:
            mat_mix[i][j] = g.matrix[i][j]

def get_key(val):
    for node,value in dict.items():
         if val == value:
             return node


def calculate_gn(rute,nodeN):
    gn = 0
    for i in range(len(rute)):
        if (i != len(rute) - 1):
            gn += mat_mix[dict[rute[i]]][dict[rute[i+1]]]
        else:
            gn += mat_mix[dict[rute[i]]][dict[nodeN]]
    return gn



arr = [] # Untuk menyimpan fn setiap kemungkinan searching
rute = [] # Menyimpan rute
```

```
def is_have_edge(node):
    int_node = dict[node]
    count = 0
    for i in range(len(mat_edge[int_node])):
        if (mat_edge[int_node][i] == 1):
            count += 1
    return count

# is_have_edge("H")

def Get_Short_Path(nodeAwal,nodeAkhir):
    iterasi = 0
    if (nodeAwal == nodeAkhir):
        print("Anda Sudah di sana :) ")
    else:
        if (is_have_edge(nodeAkhir) > 0): # Jika node akhir dapat dicapai
            a_star(nodeAwal,nodeAkhir,iterasi)
        else:
            print("Lokasi Tujuan tidak bisa dicapai")

def a_star(nodeAwal, nodeAkhir,iterasi):
    iterasi += 1
    rute.append(nodeAwal)
    if (is_have_edge(nodeAwal) == 1 and iterasi != 1):
        if (len(arr) == 0):
            print("Tidak bisa mencapai lokasi tujuan")
            rute.append("NO")
        else:
            for i in range(len(mat_mix[dict[nodeAwal]])):
                if mat_mix[dict[nodeAwal]][i] != 0:

                    nodeN = get_key(i)

                    gn = calculate_gn(rute,nodeN)
                    # print(gn)
                    hn = g.matrix[i][dict[nodeAkhir]]
                    # print(hn)
                    total = gn + hn
                    # print(total)
                    rute.append(get_key(i))
                    copy_rute = ut.copy_arr(rute)
                    arr_total = [total,copy_rute]
                    arr.append(arr_total)
                    del rute[len(rute)-1:len(rute)]

            if (len(arr) != 0):
                idx_min = ut.min_arr(arr)
                temp = arr[idx_min]
                del arr[idx_min:idx_min + 1]
                node = temp[1][len(temp[1])-1]
                if (node == nodeAkhir):
                    print("Rute Terpendek : ")
                    jarak = calculate_gn(rute,node)
                    rute.append(node)
                    print("Rute : ", end=" ")
                    # print(rute)
                    ut.display_array(rute)
```

```
                    rute.append("YES")
                    print("Iterasi : " + str(iterasi))
                    print("Jarak : " + str(jarak))

                else:
                    rute.clear() # Clear Rute

                    for i in range(len(temp[1])):
                        rute.append(temp[1][i]) # Isi Kembali Rute

                    del rute[len(rute)-1:len(rute)]

                    # recursive
                    a_star(get_key(dict[temp[1][len(temp[1])-
1]]),nodeAkhir,iterasi)
            else: # klw node awal ga punya tetangga
                print("Tidak ada rute yang dapat dilalui")
                rute.append("NO")
            # else:
            #     print("Tidak bisa mencapai lokasi tujuan Anda")
    else:
        for i in range(len(mat_mix[dict[nodeAwal]])):
            if mat_mix[dict[nodeAwal]][i] != 0:
                nodeN = get_key(i)

                gn = calculate_gn(rute, nodeN)
                # print(gn)
                hn = g.matrix[i][dict[nodeAkhir]]
                # print(hn)
                total = gn + hn
                # print(total)
                rute.append(get_key(i))
                copy_rute = ut.copy_arr(rute)
                arr_total = [total, copy_rute]
                arr.append(arr_total)
                del rute[len(rute) - 1:len(rute)]

        if (len(arr) != 0):
            idx_min = ut.min_arr(arr)
            temp = arr[idx_min]
            # print("ARR")
            # print(arr)
            # print("TEMP")
            # print(temp)
            del arr[idx_min:idx_min + 1]
            node = temp[1][len(temp[1]) - 1]
            # print(node)
            if (node == nodeAkhir):
                print("Rute Terpendek : ")
                jarak = calculate_gn(rute, node)
                rute.append(node)
                print("rute : ", end=" ")
                # print(rute)
                ut.display_array(rute)
                rute.append("YES")
                print("Iterasi : " + str(iterasi))
                print("Jarak : " + str(jarak))
```

```
            else:
                rute.clear()  # Clear Rute

                for i in range(len(temp[1])):
                    rute.append(temp[1][i])  # Isi Kembali Rute
                # print("RITEEE")
                # print(rute)
                del rute[len(rute) - 1:len(rute)]
                # print("AFTER DELETE")
                # print(rute)

                # recursive
                a_star(get_key(dict[temp[1][len(temp[1]) - 1]]), nodeAkhir,
iterasi)
        else: # klw node awal ga punya tetangga
            print("Tidak ada rute yang dapat dilalui")
            rute.append("NO")
        # else:
        #     print("Tidak bisa mencapai lokasi tujuan Anda")


axis = []
ordinat = []
for i in range(n_node):
    for j in range(n_node):
        if mat_edge[i][j] == 1:
            varx = [int_data[i][0], int_data[j][0]]
            vary = [int_data[i][1], int_data[j][1]]
            axis.append(varx)
            ordinat.append(vary)

# Menampilkan peta awal
for i in range(len(axis)):
    plt.plot(axis[i], ordinat[i], marker='o', color="blue")
for i in range(len(int_data)):
    plt.annotate(get_key(i), (int_data[i][0], int_data[i][1]))

plt.xlabel('x - axis')
plt.ylabel('y - axis')

# plt.legend()
plt.title("Map")
plt.show()

def is_in_dict(nodeInput):
    for node in dict.keys():
        if nodeInput == node:
            return True
    return False

print("List Daerah : ")
for i in range(len(arr_node)):
    print(str(i+1) + ". "+arr_node[i])

print("-----------------------------")
print("|   Dari mana mau ke mana   |")
```

```
print("|           ???             |")
print("-----------------------------")



nodeAwal = input("Input asal (Node Awal) : ")
while(not is_in_dict(nodeAwal)):
    nodeAwal = input("Ulangi input asal (Node Awal) : ")
nodeAkhir = input("Input tujuan (Node Akhir) : ")
while(not is_in_dict(nodeAkhir)):
    nodeAkhir = input("Ulangi input tujuan (Node Akhir) : ")

Get_Short_Path(nodeAwal,nodeAkhir)

# Menampilkan peta awal
for i in range(len(axis)):
    plt.plot(axis[i], ordinat[i], marker='o', color="blue")
for i in range(len(int_data)):
    plt.annotate(get_key(i), (int_data[i][0], int_data[i][1]))

# rute_axis = []
# rute_ordinat = []
# for i in range(len(rute)-1):
#     rute_axis.append(int_data[dict[rute[i]]][0])
#     rute_ordinat.append(int_data[dict[rute[i]]][1])

rute_axis = []
rute_ordinat = []
for i in range(len(rute)-1):
    rute_axis.append(int_data[dict[rute[i]]][0])
    rute_ordinat.append(int_data[dict[rute[i]]][1])
if (rute[len(rute)-1] == "YES"): # Klw sampai ke node akhir / tujuan
    plt.plot(rute_axis, rute_ordinat, label="shortest path", marker='o',
color="red")
    plt.legend()

plt.xlabel('x - axis')
plt.ylabel('y - axis')



plt.title('Map')

plt.show()
```

File graph.py

```
class Graf:
    def __init__(self, number):
```

```
        self.number = number
        self.matrix = [[0 for j in range(number)] for i in range(number)]

    def add_element(self, i, j, jarak):
        self.matrix[i][j] = jarak

    def display_matrix(self):
        for i in range(self.number):
            for j in range(self.number):
                print(self.matrix[i][j], end="\t")
            print()
```

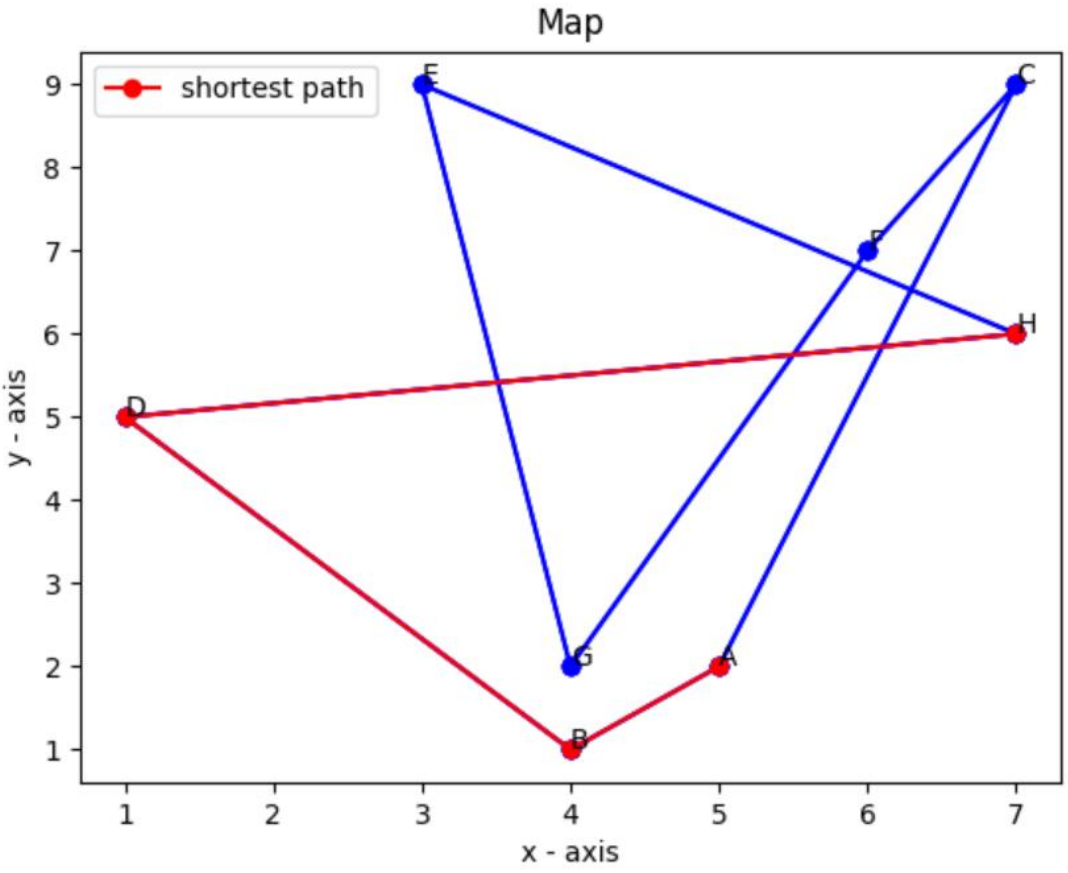
File util.py

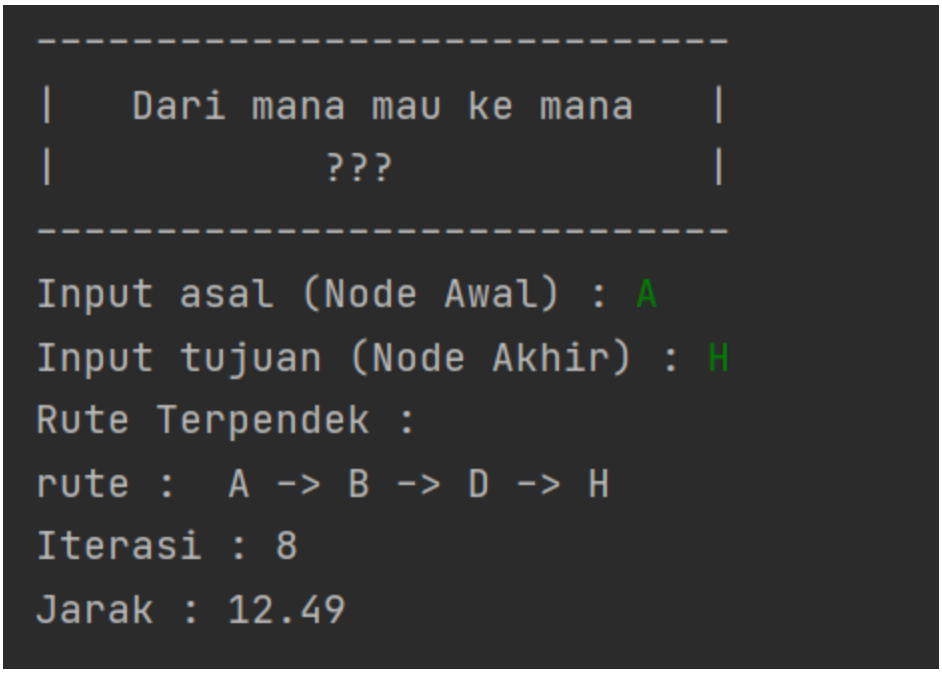
```
from math import radians, cos, sin, asin, sqrt

def Euclidean(x1, x2, y1, y2):
    hasil = ((y2-y1)**2 + (x2-x1)**2)**(0.5)
    hasil = float("{:.2f}".format(hasil))
    # hasil = int(hasil)
    # print(type(hasil))
    return(hasil)


def display_mat(matriks,n_row,n_col):
    for i in range(n_row):
        for j in range(n_col):
            print(matriks[i][j], end="\t")
        print()


def min_arr(arr_total): # return node yg minimal total dr gn + hn nya
    min = arr_total[0][0]
    idx_min = 0
    for i in range(1,len(arr_total)):
        if (arr_total[i][0] < min):
            min = arr_total[i][0]
            idx_min = i
    return idx_min

def copy_arr(arr):
    neo = []
    for i in range(len(arr)):
        neo.append(arr[i])
    return neo

def Haversine(lat1, lat2, lon1, lon2):
    lon1 = radians(lon1)
    lon2 = radians(lon2)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    dif_lon = lon2 - lon1
    dif_lat = lat2 - lat1
    a = sin(dif_lat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dif_lon / 2) ** 2
```

```
    c = 2 * asin(sqrt(a))


    r = 6371 # Jari - jari bumi dlm km

    return (c * r)

def display_array(arr):
    for i in range(len(arr)):
        if (i != len(arr)-1):
            print(arr[i], end=" -> ")
        else:
            print(arr[i])
```

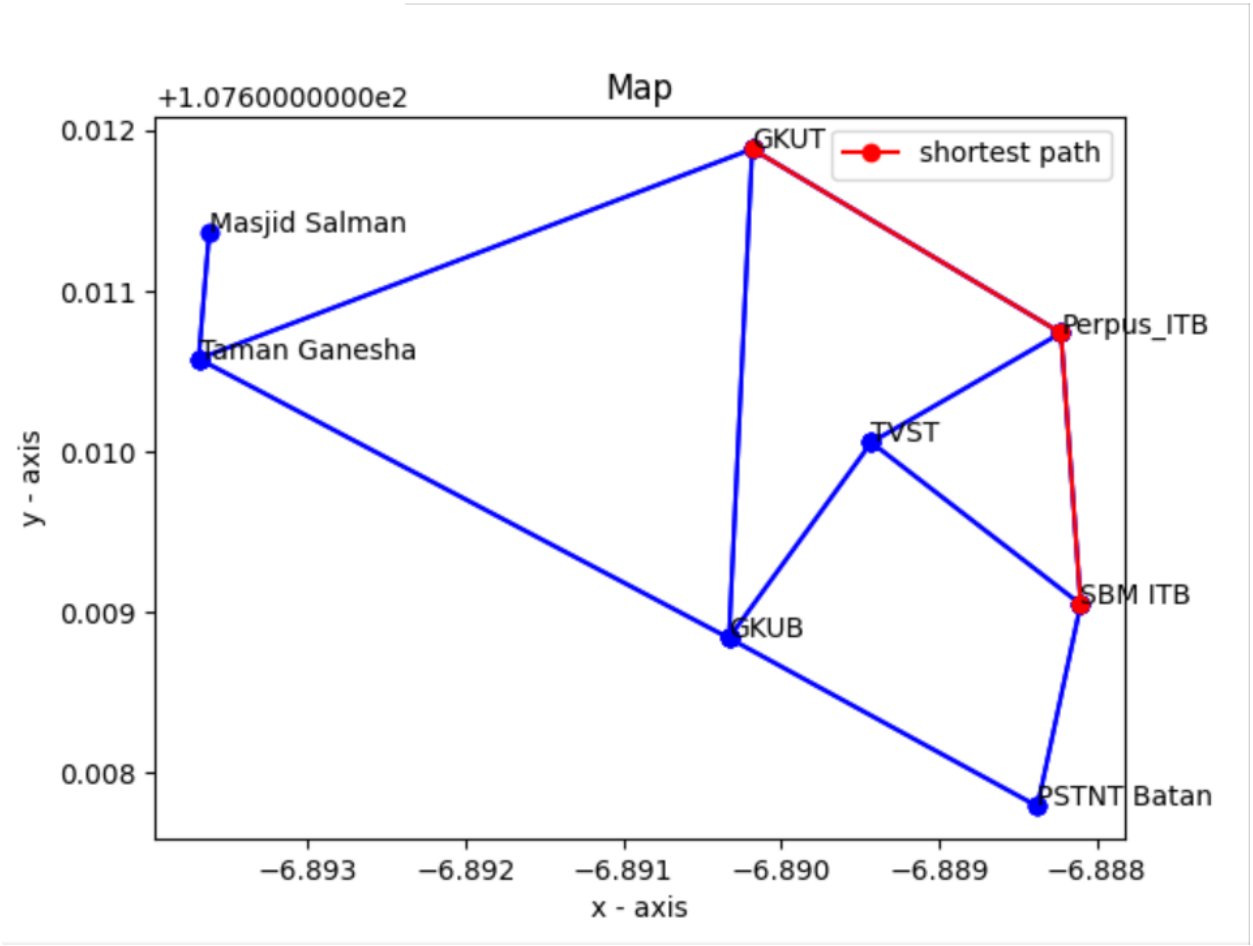
BAB IV
TEST CASE

tc1_1.txt dan tc1_2.txt (Map berdasarkan node kartesian)



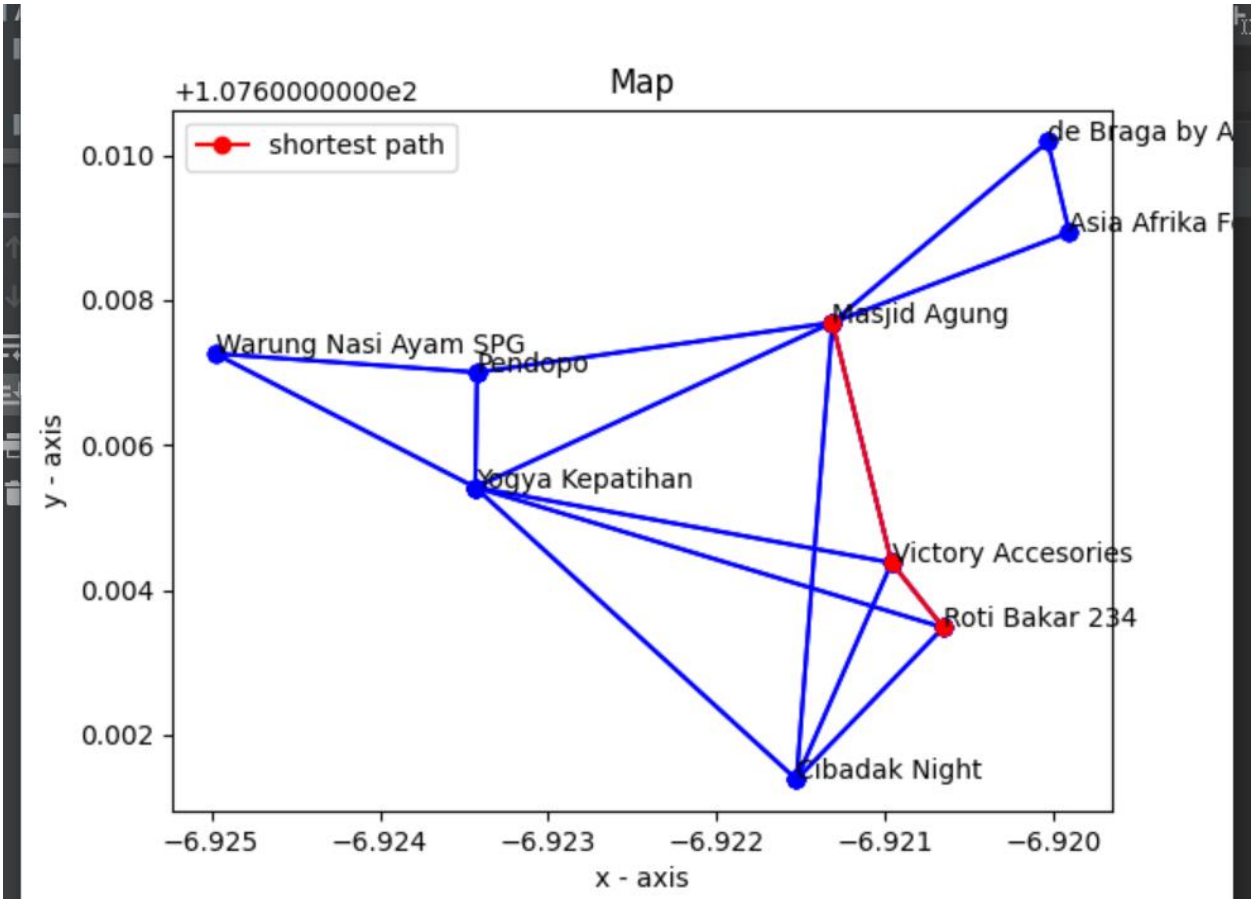


tc6_1.txt dan tc6_2.txt (Map di sekitar ITB)
Dari : GKUT
Ke : SBM ITB



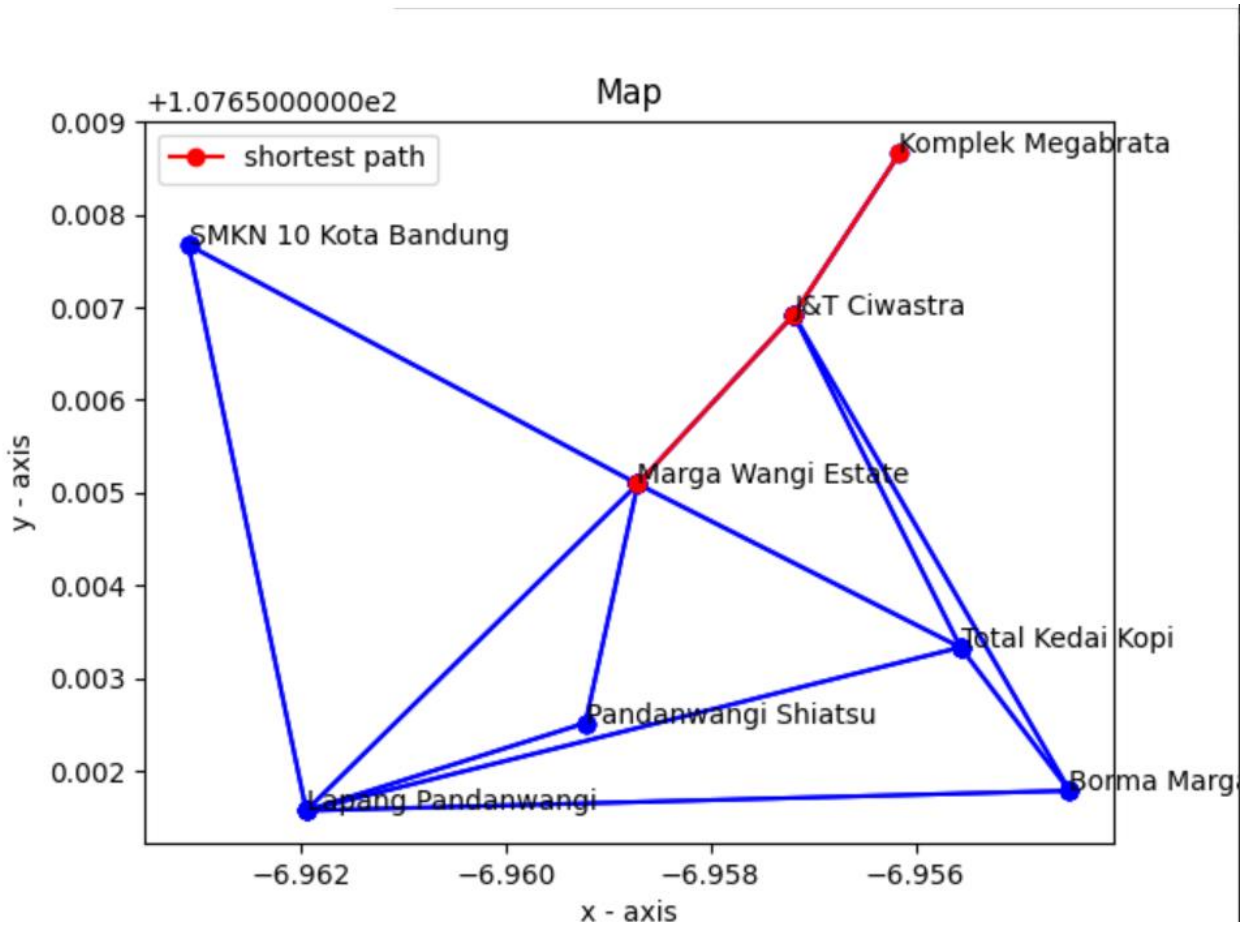

```
-----------------------------
|   Dari mana mau ke mana   |
|           ???             |
-----------------------------
Input asal (Node Awal) : GKUT
Input tujuan (Node Akhir) : SBM ITB
Rute Terpendek :
rute :  GKUT -> Perpus_ITB -> SBM ITB
Iterasi : 2
Jarak : 0.23139664234743584
```

Tc7_1.txt dan tc7_2.txt (Map di sekitar alun-alun Bandung)
Dari : Masjid Agung
Ke : Roti Bakar 234



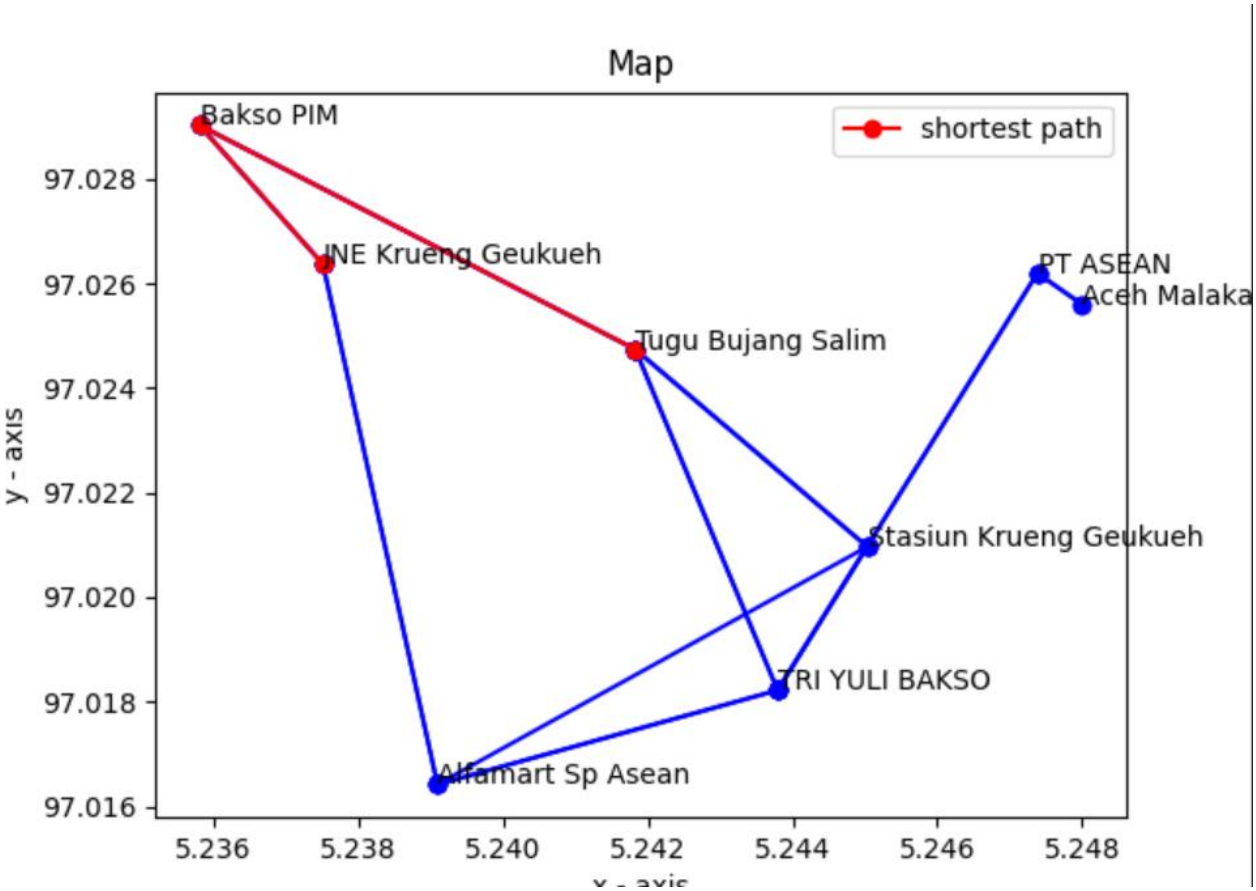

```
-----------------------------
|   Dari mana mau ke mana   |
|           ???             |
-----------------------------
Input asal (Node Awal) : Masjid Agung
Input tujuan (Node Akhir) : Roti Bakar 234
Rute Terpendek :
rute :  Masjid Agung -> Victory Accesories -> Roti Bakar 234
Iterasi : 2
Jarak : 0.07315726441347041
```

tc8_1.txt dan tc8_2.txt (Map di buah batu)
Dari : Marga Wangi Estate
Ke : Komplek Megabrata



```
-----------------------------
|   Dari mana mau ke mana   |
|           ???             |
-----------------------------
Input asal (Node Awal) : Marga Wangi Estate
Input tujuan (Node Akhir) : Komplek Megabrata
Rute Terpendek :
rute :  Marga Wangi Estate -> J&T Ciwastra -> Komplek Megabrata
Iterasi : 2
Jarak : 0.2842097812235637
```

tc9_1.txt dan tc9_2.txt (Map di sekitar tempat tinggal)
Dari : Marga Wangi Estate
Ke : Komplek Megabrata




```
-----------------------------
|   Dari mana mau ke mana   |
|           ???             |
-----------------------------
Input asal (Node Awal) : Tugu Bujang Salim
Input tujuan (Node Akhir) : JNE Krueng Geukueh
Rute Terpendek :
rute :  Tugu Bujang Salim -> Bakso PIM -> JNE Krueng Geukueh
Iterasi : 2
Jarak : 0.8601189619058969
```

Bab V
Alamat Drive Berisi Kode Program dan Tabel Hasil Program

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	

https://github.com/mfikrin/A-Star-Algorithm-Map