LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

“Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek”

OLEH :

CHRISTIAN ALEXANDRO TOBING - 13519109

CHRISTIAN GUNAWAN – 13519199

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

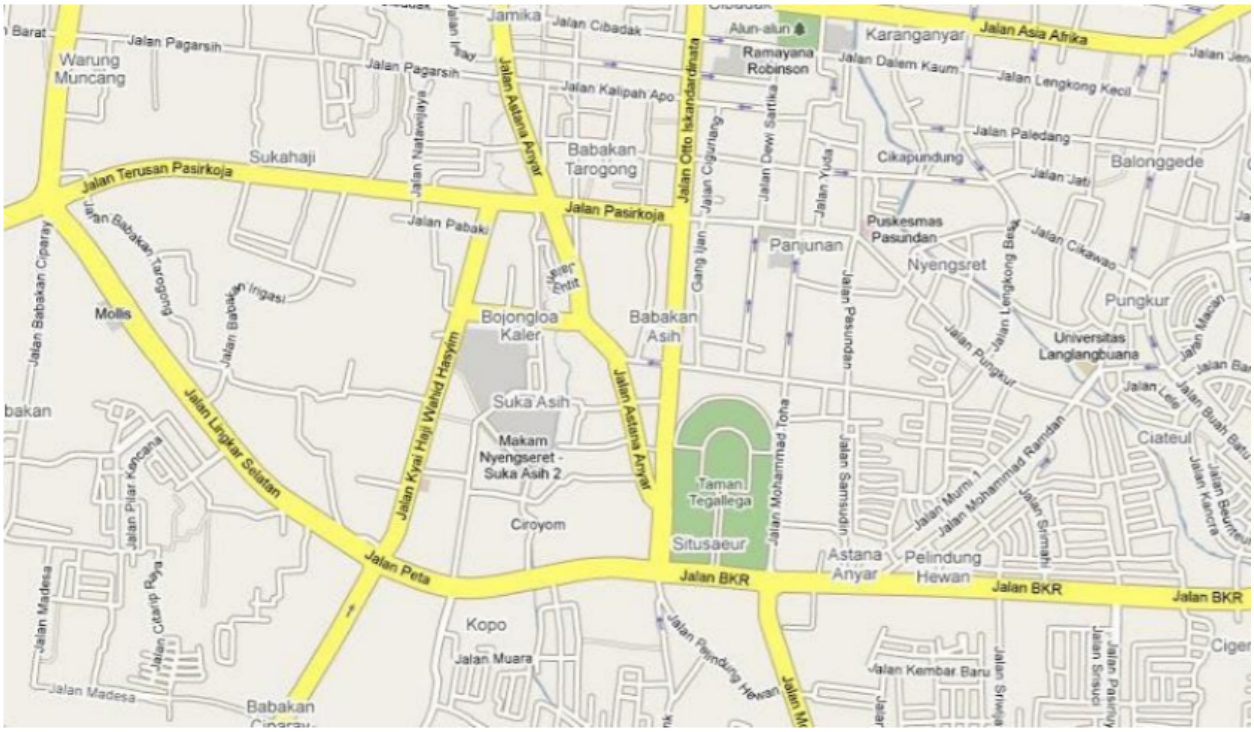
INSTITUT TEKNOLOGI BANDUNG

2021

Bab 1

Penjelasan Algoritma

Pada tugas kali ini, akan menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung dengan menggunakan algoritma A* (dibaca A star). 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 antara 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. Hasil Peta

Algoritma A* atau A star adalah salah satu algoritma yang baik dalam menemukan solusi proses *pathfinding* (pencari jalan). Algoritma ini mencari jarak rute terpendek yang akan ditempuh suatu point awal (*starting point*) sampai ke objek tujuan. Teknik pencarian yang digunakan dalam simulasi ini adalah menggunakan Algoritma A* dengan fungsi heuristic. Tujuan utama penelitian ini mempelajari cara kerja algoritma A* dalam mencari jarak tercepat, yang disimulasikan seperti kondisi ketika seorang mencari rute dalam keadaan jalanan macet. Simulasi ini memberikan gambaran yang lebih realistis terhadap perilaku algoritma A* dalam pencarian jarak rute terpendek

Bab 2

Source Code

Berikut potongan gambar dari program *pathfinder.py*

```python
from read_input import *


#total cost for nodes visited
treeDict = read_input2()[0]
def Astar(start, goal):
    heuristicDict = createHeuristicDict(goal)
    cost = {start : 0}
    """
    Input : dictionary (data structures for graph with adjacency matrix representation)
    Output : Array of nodes with A* values , Array of ordered place sequence based on shortest path using A* pathfinder algortihm
    """
    # OPEN SET
    opened = []
    # CLOSE SET
    closed = []
    # CURRENT PLACE
    current = start
    # ADD CURRENT TO OPEN
    opened.append([current, heuristicDict[current]])
    while True:
        # CHECK FOR MINIMUM HEURISTIC in OPEN SET
        if len(opened) == 0 :
            return [], [], []
        current = min(opened, key = lambda x : x[1])
        # CHECKED_NODE
        checked_node = current[0]
        # APPEND IT TO CLOSED SET
        closed.append(current)
        # AFTER APPENDING TO CLOSE, DELETE FROM OPENED
        opened.remove(current)
        # CHECK IF GOAL IS INCLUDED IN CLOSED SET
        if(closed[-1][0] == goal):
            break
        # IF THE PREVIOUS CONDITIONAL GUARD IS NOT FULFILLED, move to the next children's node
        for children in treeDict[checked_node].items():
            # CHECK IF CHILDREN NODES ALREADY IN CLOSED SET
            if children[0] in [closed_nodes[0] for closed_nodes in closed]:
                continue
            # UPDATE THE CHILDREN PATH WITH RECENT VALUE, guaranteed minimum because of lambda minimum function in line 24
            cost.update({children[0] : cost[checked_node] + children[1]})
            # CAST F VALUE (HEURISTIC + CURRENT PATH COST) OF CURRENT NODES
            current_fval = cost[checked_node] + heuristicDict[children[0]] + children[1]
            # ADD TO OPENED AFTER COUNTING THE F VALUES
            # USED TEMP SO THE ACTION WON'T CORRUPT THE treeDict
            temp = [children[0], current_fval]
            opened.append(temp)
    ''' ordering the sequence based on A* values
    e.g of A* closed set = [['ITB', 65], ['DagoA', 70], ['DagoB', 70], ['DagoC', 70], ['Goal', 75]] '''
    # DEFINING THE LAST_NODE OR GOAL_NODE
    last_node = goal
    ordered_sequence = []
    ordered_sequence.append(goal)
    # TRACK FROM lastindex-1 to the first index
    for i in range(len(closed) - 2, -1, -1):
        # DEFINE CHECKED NODE AS STRING
        check_node = closed[i][0]
        # CHECK IF THE GOAL IS THE CHILD OF THE CHECKED NODE
        if last_node in [children[0] for children in treeDict[check_node].items()]:
            if (cost[check_node] + treeDict[check_node][last_node] == cost[last_node]):
                ordered_sequence.append(check_node)
                last_node = check_node
    # Reverse ordering from ordered_sequence
    ordered_sequence.reverse()
    return closed, ordered_sequence, cost

def createHeuristicDict(goal):
    heuristicDict2 = dict()
    for nodes in treeDict:
        heuristicDict2[nodes] = euclideanDistance(nodes, goal)
    return heuristicDict2

```

Berikut potongan gambar dari program *graph.py*

```python
import networkx as nx
import matplotlib.pyplot as plt
from read_input import coordinateDict
from PathFinder import Astar, treeDict


def graphMaker(start, goal):
    # Inisialisasi Graph
    G = nx.Graph()
    # Initialize variables from PathFinder
    visited_graph, ordered_sequence, cost_function = Astar(start, goal)
    # Jarak
    distance_sum = 0
    for i in range(len(ordered_sequence) - 1):
        distance_sum += treeDict[ordered_sequence[i]][ordered_sequence[i + 1]]
    # Bentuk semua nodes
    for nodes in treeDict:
        G.add_node(nodes, pos = (coordinateDict[nodes]['lat'], coordinateDict[nodes]['lng']))
    # Bentuk semua edges
    for nodes in treeDict:
        for children in treeDict[nodes]:
            G.add_edge(nodes, children, weight = treeDict[nodes][children])

    # Posisi
    pos = nx.get_node_attributes(G, 'pos')
    # Bobot
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G,pos, edge_labels= labels)
    # Mewarnai node yang dikunjungi
    node_color = []
    for node in G.nodes:
        if node in ordered_sequence:
            node_color.append("red")
        else :
            node_color.append("blue")

    nx.draw(G, pos , with_labels = True, node_size = 1200,  node_color = node_color)
    for i in range(len(ordered_sequence)):
        if(i != len(ordered_sequence) - 1):
            print(f"{ordered_sequence[i]} =>", end = " ")
        else:
            print(ordered_sequence[i])
    if distance_sum == 0 :
        print("Jalur tidak ditemukan atau menunjuk ke diri sendiri")
    else :
        print(f"Panjang lintasan adalah {distance_sum}")
    plt.show()
```

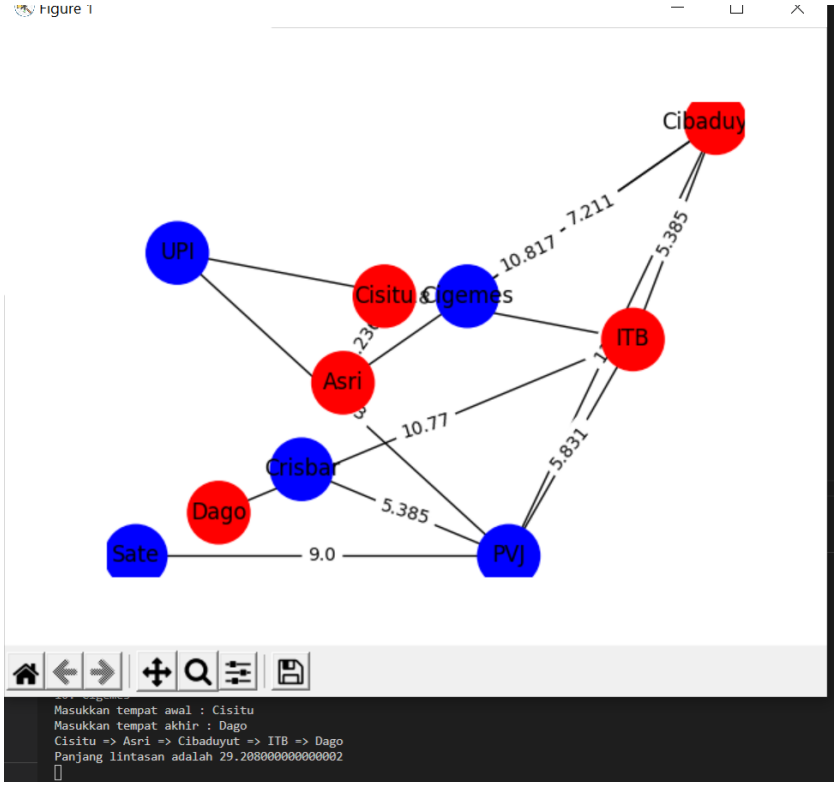
Berikut potongan gambar dari program *Main.py*

```python
from read_input import *
from PathFinder import *
from graph import *

status = True
while status:
    a = treeDict
    listTempat = [nodes for nodes in a]
    # List nama tempat
    print("LIST NAMA TEMPAT")
    num = 1
    for i in listTempat:
        print(f"{num}. {i}")
        num += 1
    # Input tempat awal
    string_awal = str(input("Masukkan tempat awal : "))
    while(string_awal not in listTempat):
        print("Salah memasukkan nama tempat awal")
        string_awal = str(input("Masukkan tempat awal : "))
    # Input tempat akhir
    string_akhir = str(input("Masukkan tempat akhir : "))
    while(string_akhir not in listTempat):
        print("Salah memasukkan nama tempat akhir")
        string_akhir = str(input("Masukkan tempat awal : "))
    # Membentuk graph
    graphMaker(string_awal, string_akhir)

    repeat = str(input("Mau mencoba dengan test-case lain ? : (Y/N): "))
    if(repeat == 'Y' or status == 'y'):
        a = treeDict
    elif(repeat =="N" or repeat == "n"):
        status = False
```

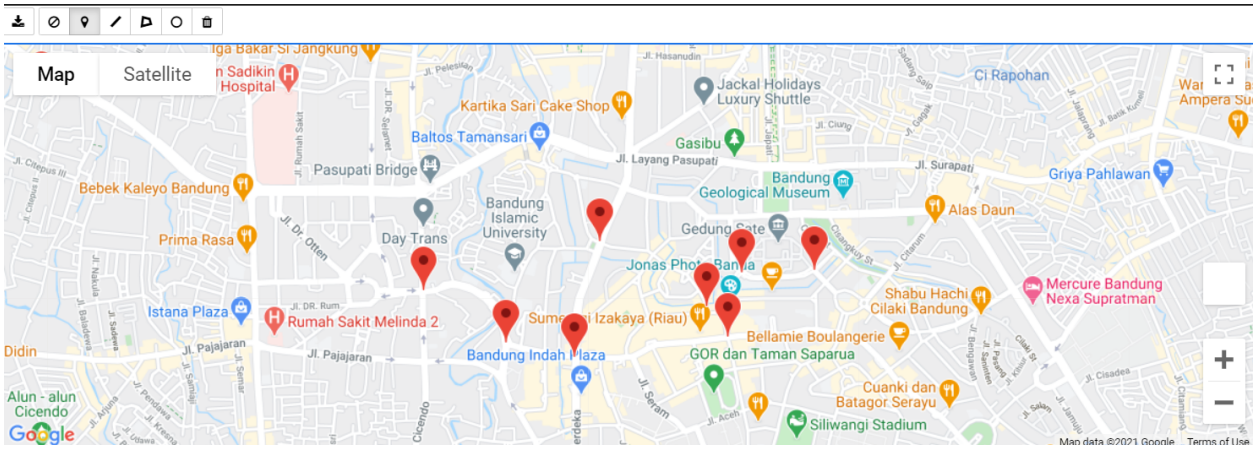
Bab 3

Hasil Percobaan

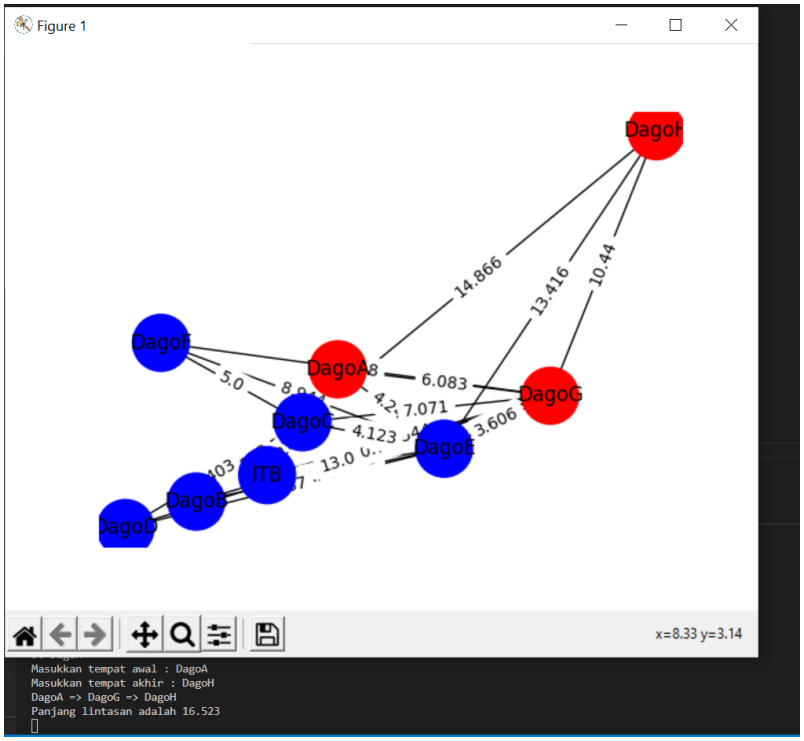
File tes1 : Hasil pada program



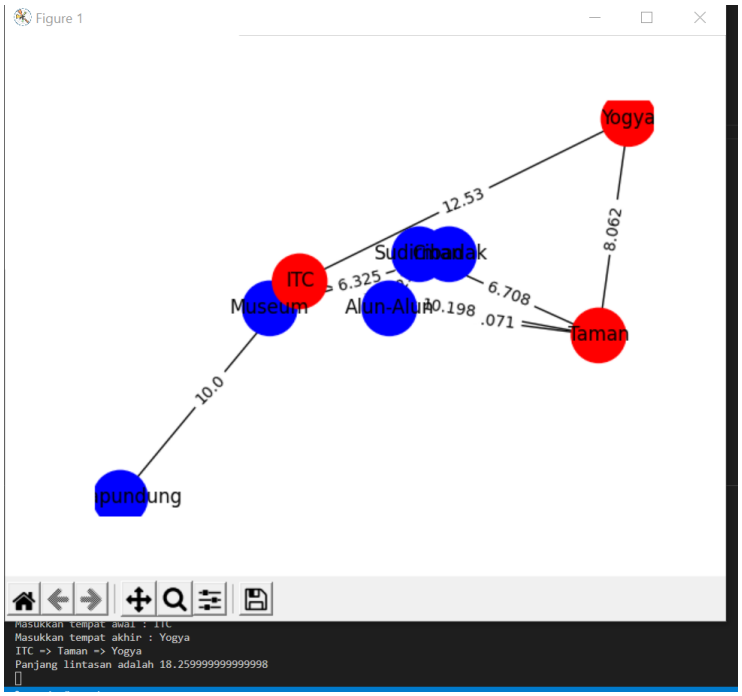
Hasil pada Google API :



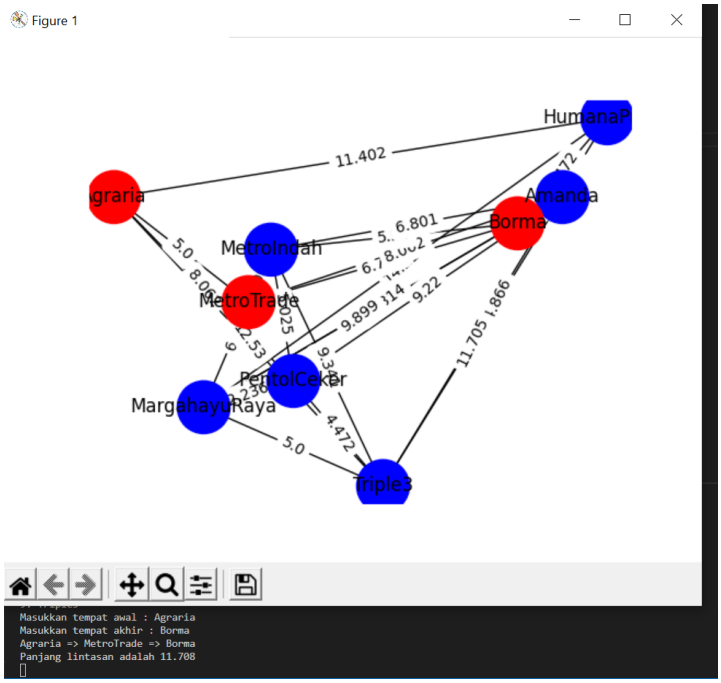
File tes2 :



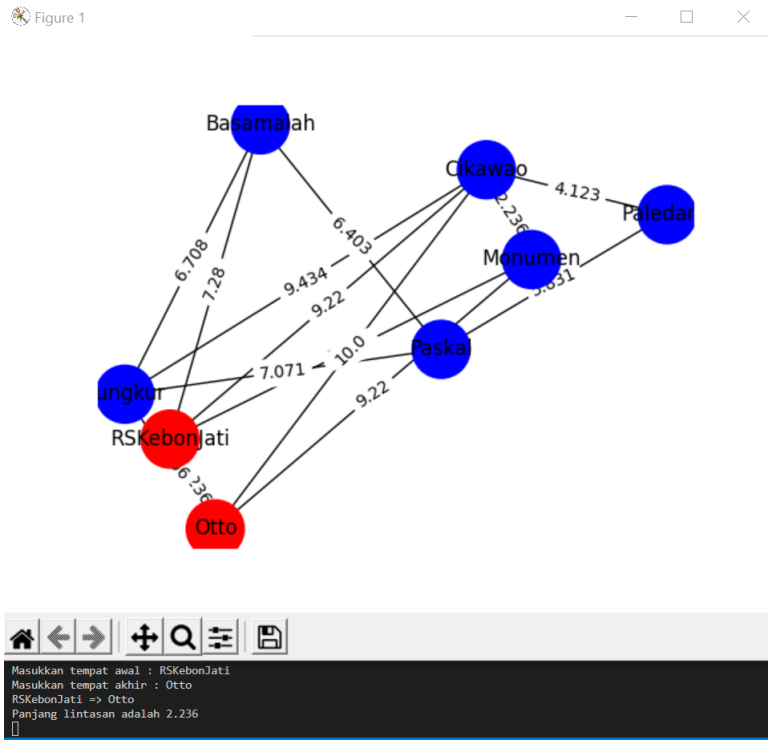
File tes3 :



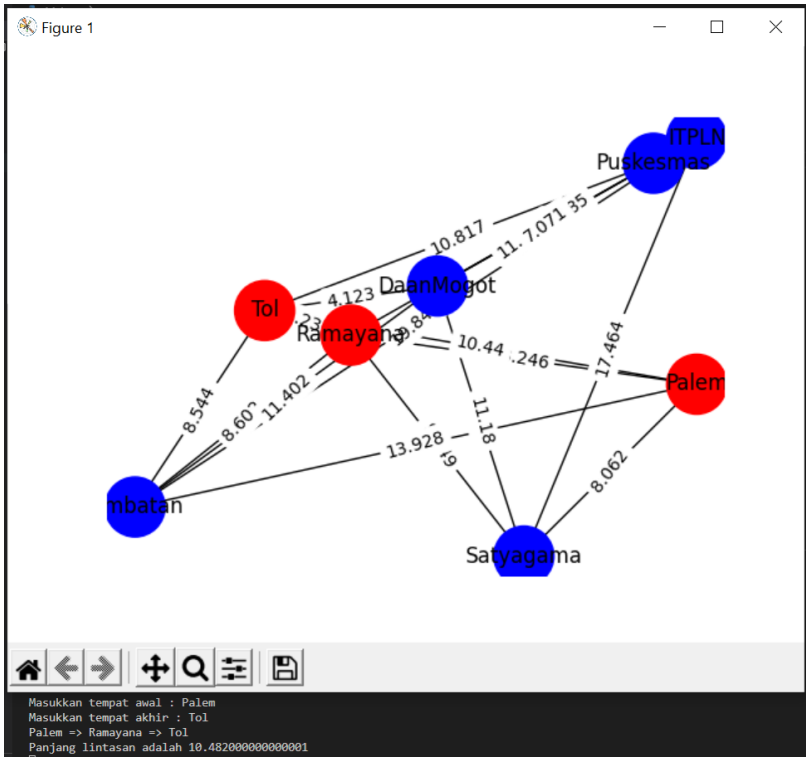
File tes4 :



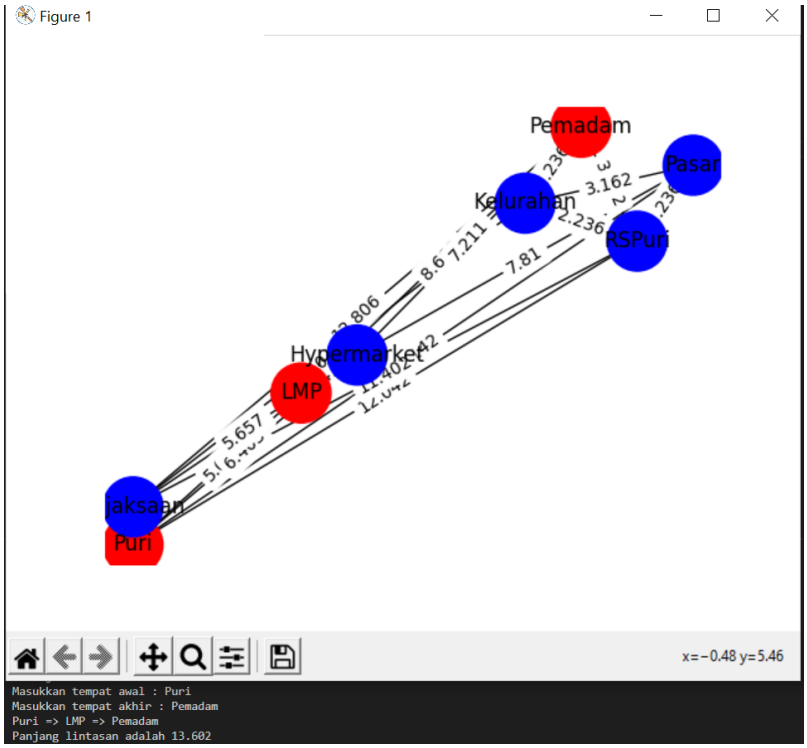
File tes5 :



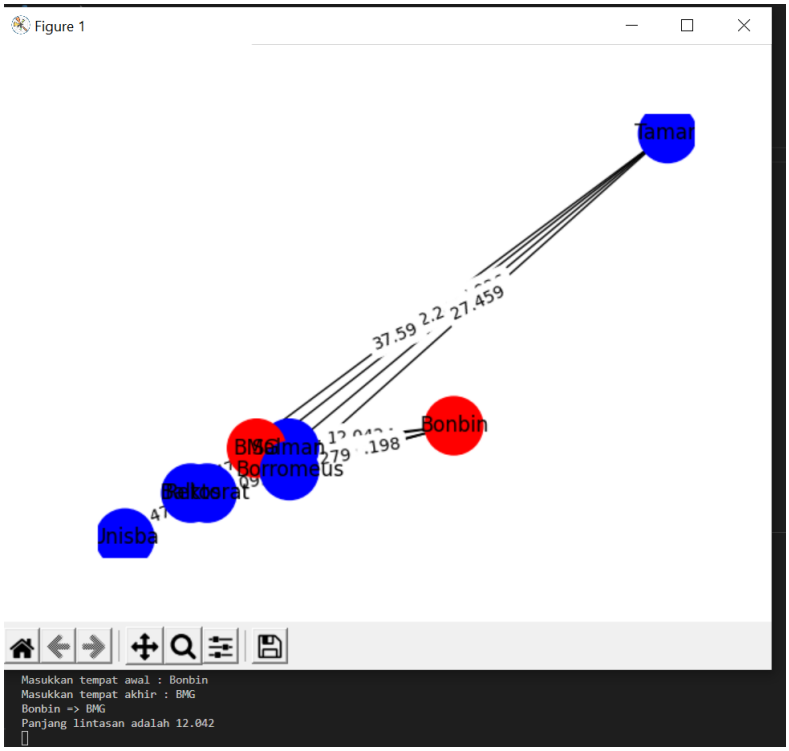
File tes6 :



File tes7 :



File tes8 :



LINK SOURCE CODE : https://github.com/chrslex/Astar-PathFinder

1	Program dapat menerima input graf	**v**
2	Program dapat menghitung lintasan terpendek	**v**
3	Program dapat menampilkan lintasan terpendek serta jaraknya	**v**
4	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta	**v**