LAPORAN TUGAS KECIL 3

IF2211/Strategi Algoritma
Semester II Tahun 2020/2021

Dipersiapkan oleh:

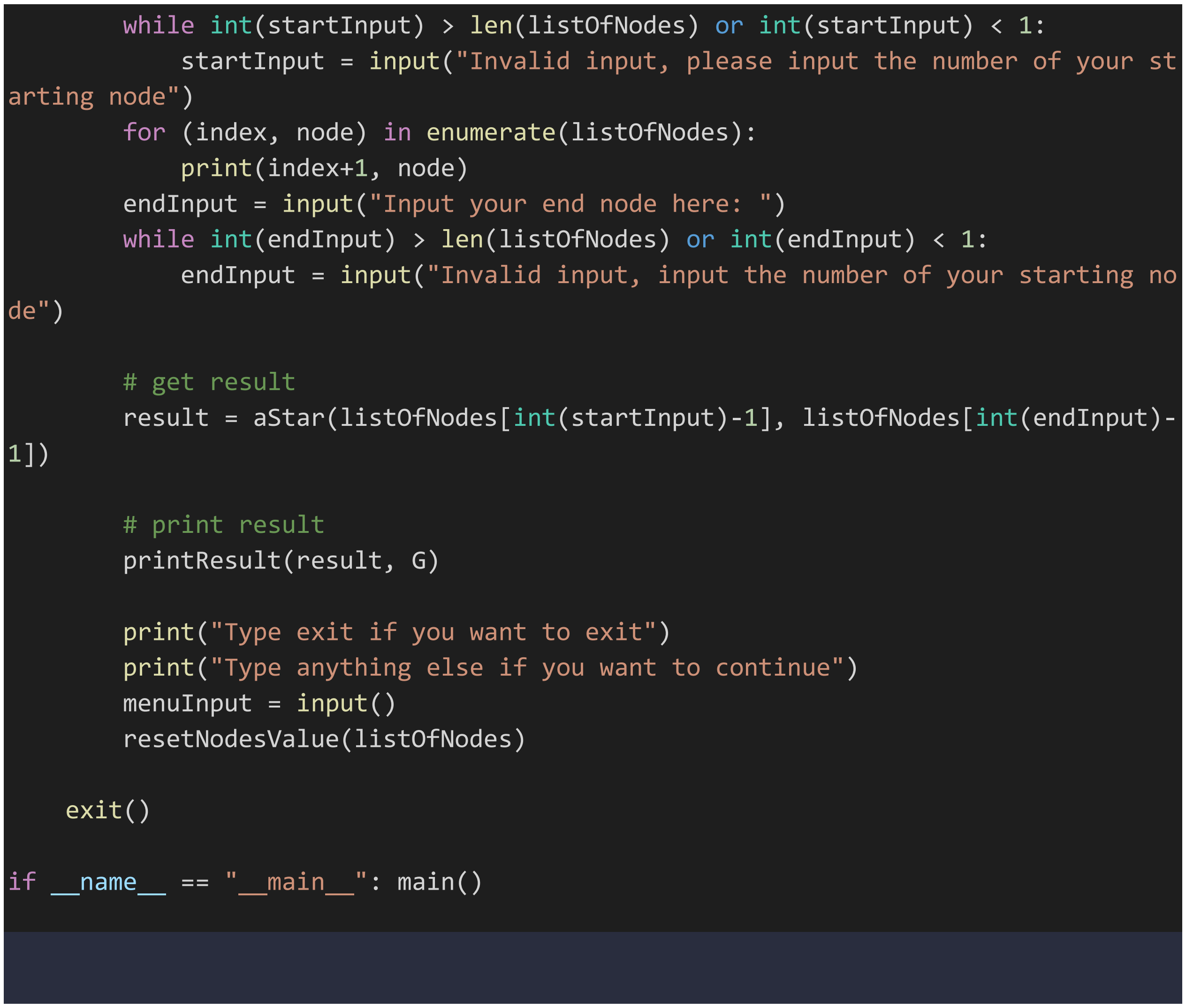
Fakhri Nail Wibowo	13519035
Dwianditya Hanif Raharjanto	13519046

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Source Code

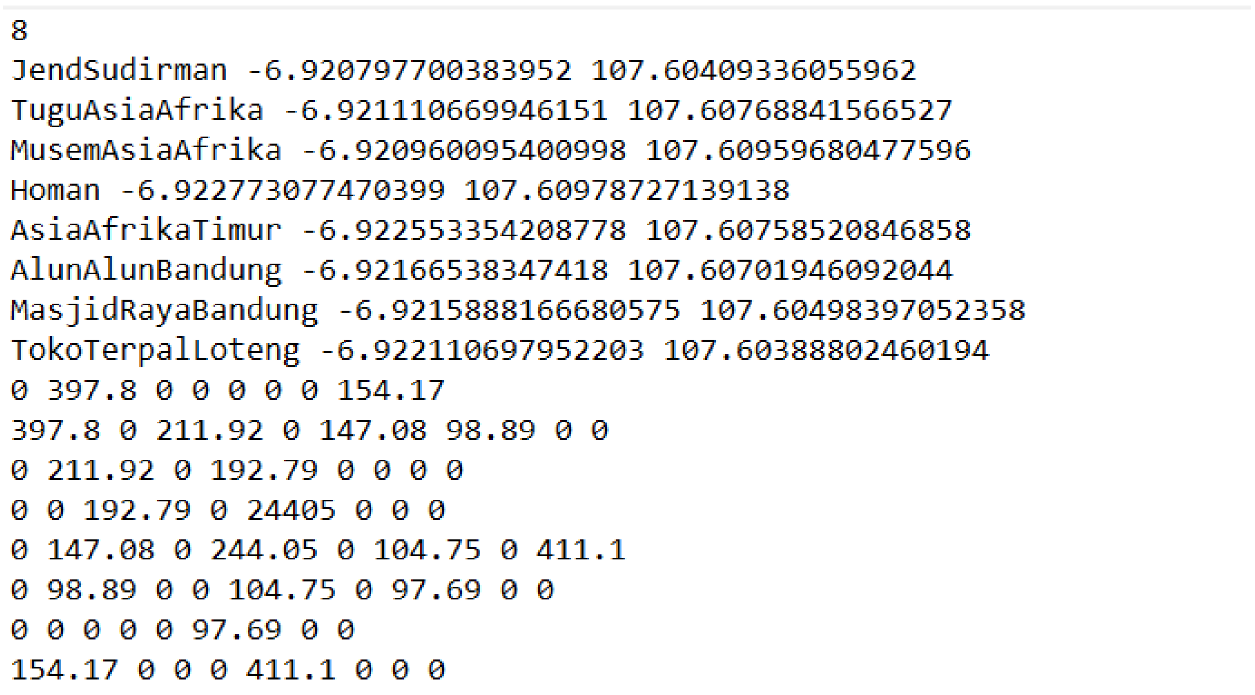
```python
import math
import networkx as nx
import matplotlib.pyplot as plt

class Node:
    def __init__(self, name, x, y):
        self.name = name
        self.positionX = x
        self.positionY = y
        self.parent = None
        self.g = 0 # traversal distance from starting node
        self.h = 0 # euclidan distance from end node
        self.f = 0 # g + h
        self.neighboringNodes = {} # dict of neighboring nodes and edges' weight

    # hash overload
    def __hash__(self):
        return hash(self.name)

    # == overload
    def __eq__(self, value):
        return self.name == value.name

    # < overload
    def __lt__(self, other):
        if self.f == other.f:
            return self.g < other.g

        return self.f < other.f

    # print overload
    def __repr__(self):
        return self.name

    # calculate euclidean distance between nodes
    def calculateEuclideanDistance(self, other):
        return math.sqrt(pow(self.positionX-
other.positionX, 2) + pow(self.positionY-other.positionY, 2))

    # calculate G value between parent and node
    def calculateG(self, parent):
        return self.neighboringNodes[parent] + parent.getG()
    # reset node a* values
    def resetValues(self):
        self.parent = None
        self.g = 0
        self.h = 0
        self.f = 0

    def setParent(self, parent):
        self.parent = parent

    def setH(self, valueH):
        self.h = valueH

    def setG(self, valueG):
        self.g = valueG

    def setF(self, valueF):
        self.f = valueF

    def setNeighboringNodes(self, neighboringNodes):
        self.neighboringNodes = neighboringNodes

    def addEdge(self, edgeNode, edgeValue):
            self.neighboringNodes[edgeNode] = edgeValue

    def getName(self):
        return self.name

    def getX(self):
        return self.positionX

    def getY(self):
        return self.positionY

    def getParent(self):
        return self.parent

    def getG(self):
        return self.g

    def getH(self):
        return self.h

    def getF(self):
        return self.f
    def getNeighboringNodes(self):
        return self.neighboringNodes

# draw initial graph
def drawGraph(G):
    pos = nx.get_node_attributes(G, 'pos')
    #labels
    labels = nx.get_edge_attributes(G, 'weight')

    #biar ada weight
    nx.draw_networkx_edge_labels(G,pos,edge_labels=labels,font_size=8)
    nx.draw(G,pos,font_size=10,node_size=500,node_color='red',edge_color='red',wi
dth=5, with_labels = True)
    plt.show()
    #End of Visualisasi Graph awal

# draw result graph
def drawResult(G,resultGraph):

    #position
    pos = nx.get_node_attributes(G, 'pos')
    #labels
    labels = nx.get_edge_attributes(G, 'weight')

    #biar ada weight
    nx.draw_networkx_edge_labels(G,pos,edge_labels=labels,font_size=8)


    # Set all edge color attribute to red
    for e in G.edges():
        G[e[0]][e[1]]['color'] = 'red'
    # Set color of edges of the shortest path to yellow
    for i in range(len(resultGraph)-1):
        G[resultGraph[i]][resultGraph[i+1]]['color'] = 'yellow'
    # Store in a list to use for drawing
    edge_color_list = [ G[e[0]][e[1]]['color'] for e in G.edges() ]
    node_colors = ["yellow" if n in resultGraph else "red" for n in G.nodes()]
    nx.draw(G,pos,font_size=10,node_size=500,node_color=node_colors,edge_color =
edge_color_list,width=5, with_labels = True)
    plt.show()

# reset all nodes a* values
def resetNodesValue(listOfNodes):
    for node in listOfNodes:
        node.resetValues()

# init nodes and edge
def initNodesAndEdges(rawNodes, adjMatrix, countNodes):
    listOfNodes =  []

    # init nodes
    for i in range(countNodes):
        nodeName = rawNodes[i][0]
        nodePositionX = float(rawNodes[i][1])
        nodePositionY = float(rawNodes[i][2])
        listOfNodes.append(Node(nodeName, nodePositionX, nodePositionY))

    # add edges to nodes
    for i in range(len(adjMatrix)):
        for j in range(len(adjMatrix)):
            if j != i and float(adjMatrix[i][j]) > 0:
                listOfNodes[i].addEdge(listOfNodes[j], float(adjMatrix[i][j]))

    return listOfNodes

# a* algorithm
def aStar(startNode, endNode):
    print("Start node is", startNode)
    print("End node is", endNode)
    openQueue = [] # priority queue for to-be-evaluated-nodes
    finishedList = [] #  list for already evaluated nodes
    result = [] # result path
    shortestDistance = 0

    # start algorithm
    openQueue.append(startNode)
    while len(openQueue) != 0:
        # sort based on f value
        openQueue.sort()
        # pop node with the least f value
        currentNode = openQueue.pop(0)
        finishedList.append(currentNode)

        # target node reached
        if currentNode == endNode:
            shortestDistance = currentNode.getG()
            # retrace path
            while currentNode != startNode:
                result.append(currentNode)
                currentNode = currentNode.getParent()

            result.append(startNode)
            result.reverse()

            # return path and its distance
            return (result, shortestDistance)

        listOfNeighboringNodes = currentNode.getNeighboringNodes().keys()
        for neighboringNode in listOfNeighboringNodes:
            if neighboringNode in finishedList:
                continue

            # calculate f,g,h
            newG = neighboringNode.calculateG(currentNode)
            newH = neighboringNode.calculateEuclideanDistance(endNode)
            newF = newH + newG
            if newG < neighboringNode.getG() or neighboringNode not in openQueue:
                neighboringNode.setG(newG)
                neighboringNode.setH(newH)
                neighboringNode.setF(newF)
                neighboringNode.setParent(currentNode)
                if neighboringNode not in openQueue:
                    openQueue.append(neighboringNode)

def printResult(result, graph):
    if result is None:
        print("No available path")
    else:
        printPath(result[0])
        printDistance(result[1])

        # add result nodes to list
        resultGraph=[]
        for node in result[0]:
            resultGraph.append(node.name)

        #Visualize path
        drawResult(graph, resultGraph)

def printPath(path):
    for (i, node) in enumerate(path):
        if i == len(path)-1:
            print(node)
        else:
            print(node , "->", end=" ")

def printDistance(distance):
    print("The shortest distance is " + str(distance))

def main():
    # get input file
    fileName = input("Input your file name: ")
    file = open("../test/" + fileName + ".txt", "r")
    lines = file.readlines()
    rawNodes = []
    adjMatrix = []

    # read input file
    countNodes = int(lines[0])
    for i in range(1, len(lines)):
        if i <= countNodes:
            rawNodes.append(lines[i].split())
        else:
            adjMatrix.append(lines[i].split())

    listOfNodes = initNodesAndEdges(rawNodes, adjMatrix, countNodes)
    print("Welcome to our map!")

    #Visualisasi Graph
    G=nx.Graph()
    for i in range(countNodes):
        G.add_node(rawNodes[i][0],pos=(float(rawNodes[i][1]),float(rawNodes[i][2]
)) )

    for i in range(len(adjMatrix)):
        for j in range(len(adjMatrix)):
            if j != i and float(adjMatrix[i][j]) > 0:
                G.add_edge(rawNodes[i][0],rawNodes[j][0],weight=float(adjMatrix[i
][j]))

    # draw initial graph
    drawGraph(G)

    menuInput = "continue"
    while menuInput != "exit":
        # ask input
        for (index, node) in enumerate(listOfNodes):
            print(index+1, node)
        startInput = input("Input your starting node here: ")
```



```
        while int(startInput) > len(listOfNodes) or int(startInput) < 1:
            startInput = input("Invalid input, please input the number of your st
arting node")
        for (index, node) in enumerate(listOfNodes):
            print(index+1, node)
        endInput = input("Input your end node here: ")
        while int(endInput) > len(listOfNodes) or int(endInput) < 1:
            endInput = input("Invalid input, input the number of your starting no
de")

        # get result
        result = aStar(listOfNodes[int(startInput)-1], listOfNodes[int(endInput)-
1])

        # print result
        printResult(result, G)

        print("Type exit if you want to exit")
        print("Type anything else if you want to continue")
        menuInput = input()
        resetNodesValue(listOfNodes)

    exit()

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

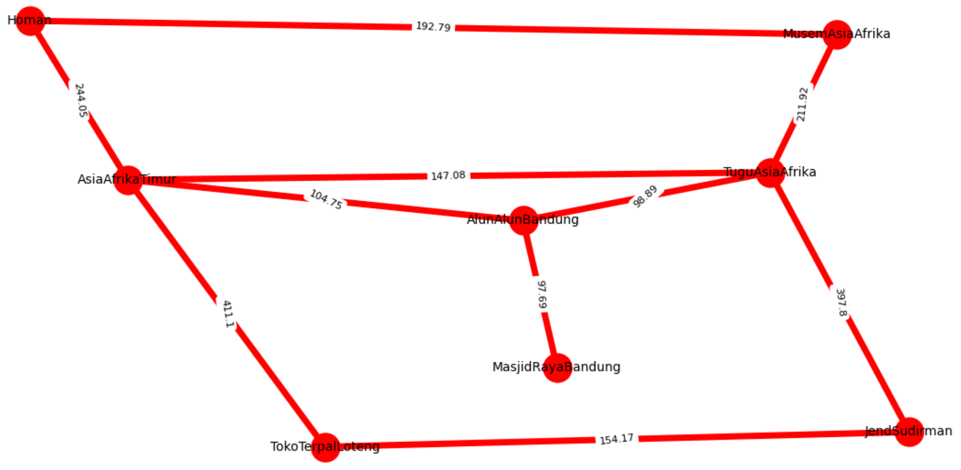
Input Graf dan Hasilnya

Alun2Bandung.txt



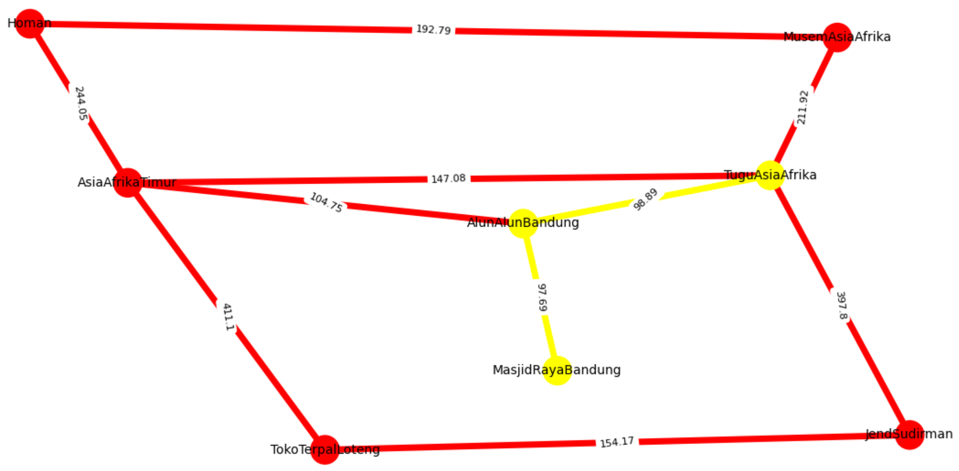
```
8
JendSudirman -6.920797700383952 107.60409336055962
TuguAsiaAfrika -6.921110669946151 107.60768841566527
MusemAsiaAfrika -6.920960095400998 107.60959680477596
Homan -6.922773077470399 107.60978727139138
AsiaAfrikaTimur -6.922553354208778 107.60758520846858
AlunAlunBandung -6.92166538347418 107.60701946092044
MasjidRayaBandung -6.9215888166680575 107.60498397052358
TokoTerpalLoteng -6.922110697952203 107.60388802460194
0 397.8 0 0 0 0 0 154.17
397.8 0 211.92 0 147.08 98.89 0 0
0 211.92 0 192.79 0 0 0 0
0 0 192.79 0 24405 0 0 0
0 147.08 0 244.05 0 104.75 0 411.1
0 98.89 0 0 104.75 0 97.69 0 0
0 0 0 0 0 97.69 0 0
154.17 0 0 0 411.1 0 0 0
```

Visualisasi Graf

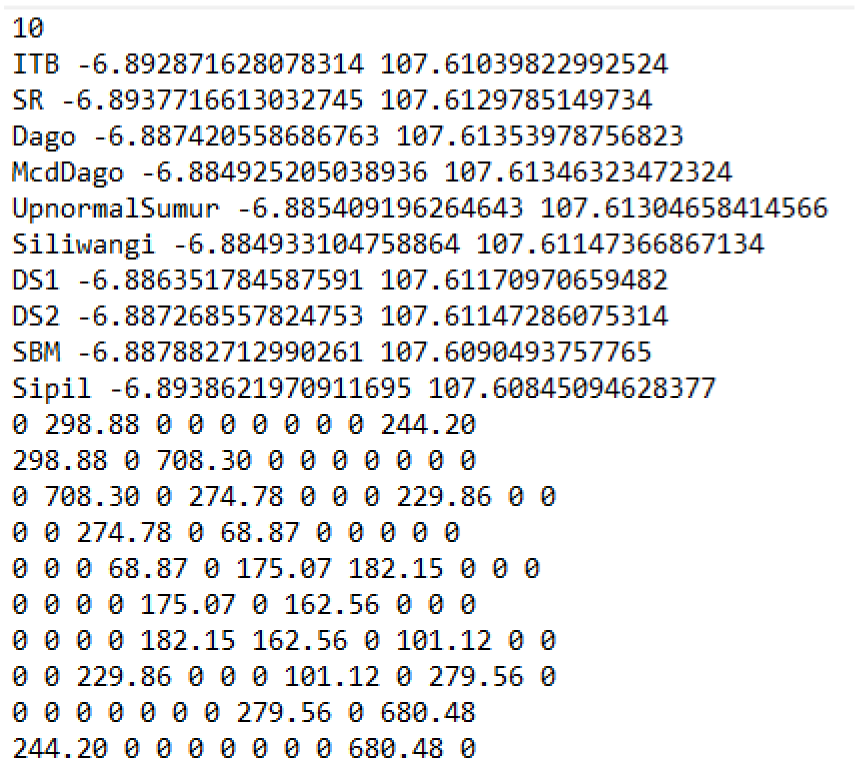


Contoh shortest path

TuguAsiaAfrika ke MasjidRayaBandung

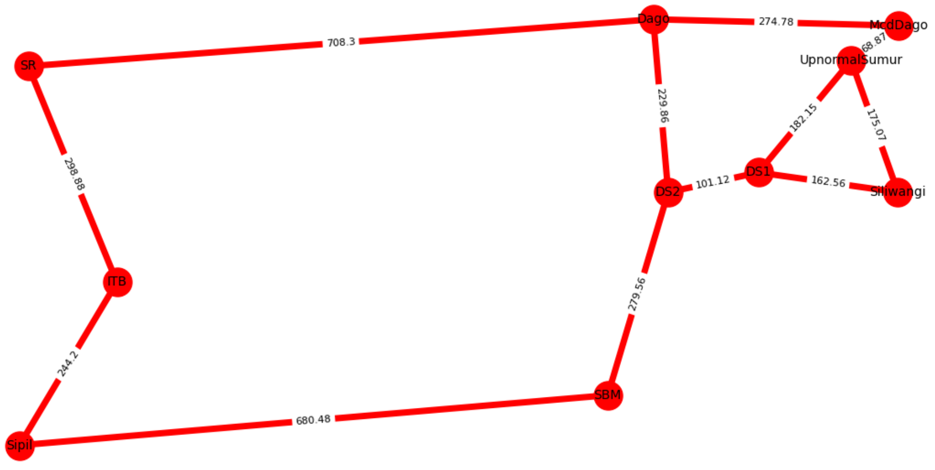


ITB.txt

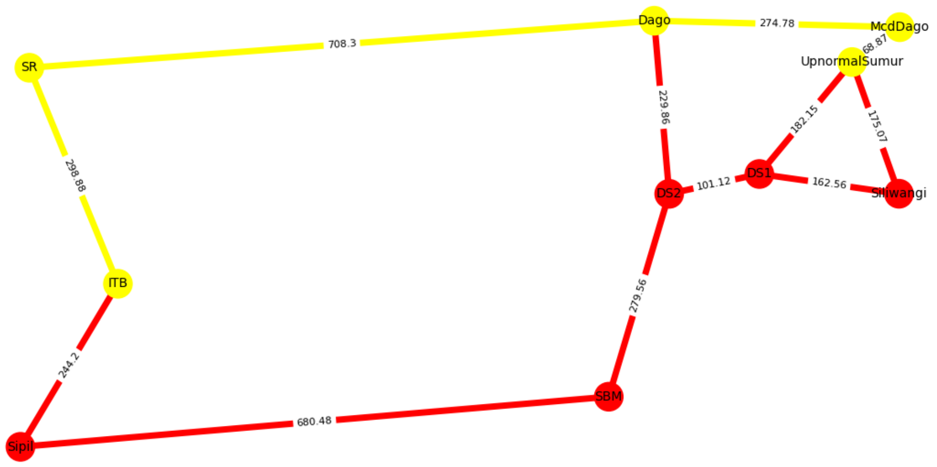


```
10
ITB -6.892871628078314 107.61039822992524
SR -6.8937716613032745 107.6129785149734
Dago -6.887420558686763 107.61353978756823
McdDago -6.884925205038936 107.61346323472324
UpnormalSumur -6.885409196264643 107.61304658414566
Siliwangi -6.884933104758864 107.61147366867134
DS1 -6.886351784587591 107.61170970659482
DS2 -6.887268557824753 107.61147286075314
SBM -6.887882712990261 107.6090493757765
Sipil -6.8938621970911695 107.60845094628377
0 298.88 0 0 0 0 0 0 0 244.20
298.88 0 708.30 0 0 0 0 0 0 0
0 708.30 0 274.78 0 0 0 229.86 0 0
0 0 274.78 0 68.87 0 0 0 0 0
0 0 0 68.87 0 175.07 182.15 0 0 0
0 0 0 0 175.07 0 162.56 0 0 0
0 0 0 0 182.15 162.56 0 101.12 0 0
0 0 229.86 0 0 0 101.12 0 279.56 0
0 0 0 0 0 0 0 279.56 0 680.48
244.20 0 0 0 0 0 0 0 680.48 0
```

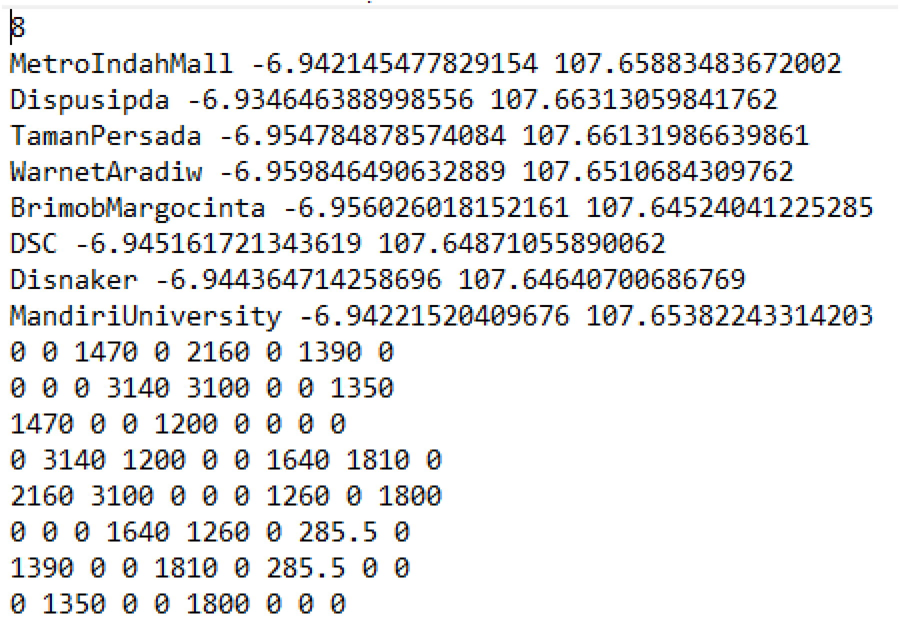
Visualisasi Graf



Contoh shortest path
ITB ke UpnormalSumur

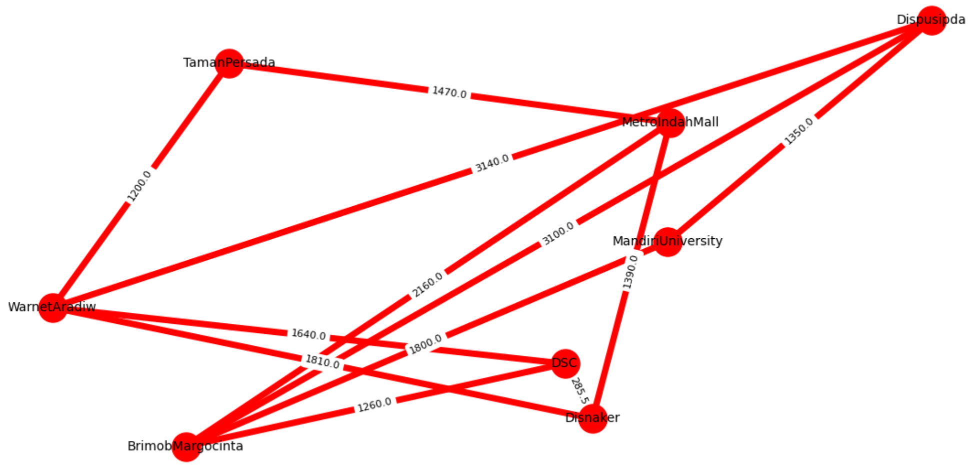


BuahBatu.txt

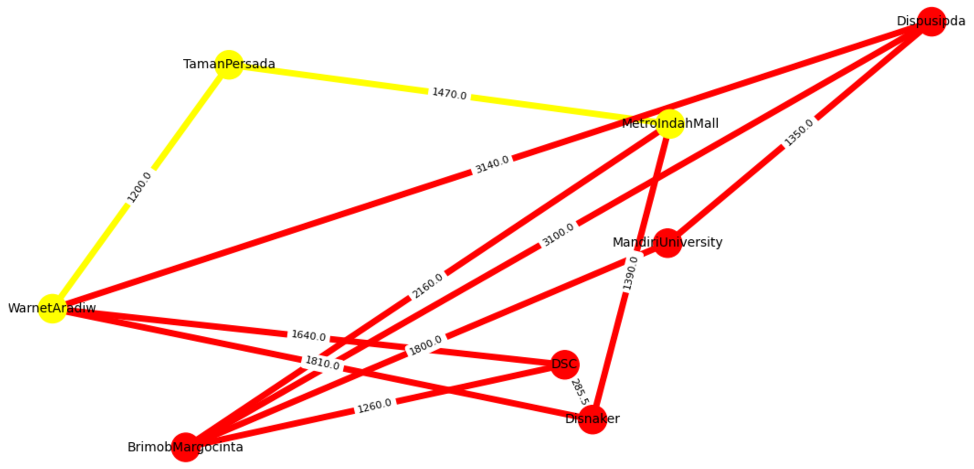


```
8
MetroIndahMall -6.942145477829154 107.65883483672002
Dispusipda -6.934646388998556 107.66313059841762
TamanPersada -6.954784878574084 107.66131986639861
WarnetAradiw -6.959846490632889 107.6510684309762
BrimobMargocinta -6.956026018152161 107.64524041225285
DSC -6.945161721343619 107.64871055890062
Disnaker -6.944364714258696 107.64640700686769
MandiriUniversity -6.94221520409676 107.65382243314203
0 0 1470 0 2160 0 1390 0
0 0 0 3140 3100 0 0 1350
1470 0 0 1200 0 0 0 0
0 3140 1200 0 0 1640 1810 0
2160 3100 0 0 0 1260 0 1800
0 0 0 1640 1260 0 285.5 0
1390 0 0 1810 0 285.5 0 0
0 1350 0 0 1800 0 0 0
```

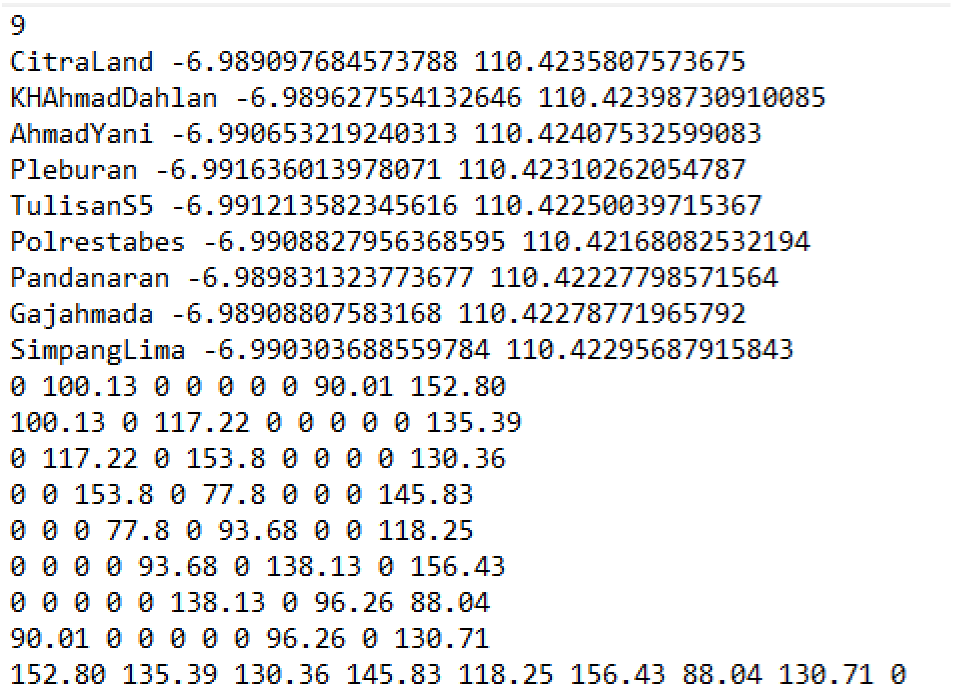
Visualisasi Graph



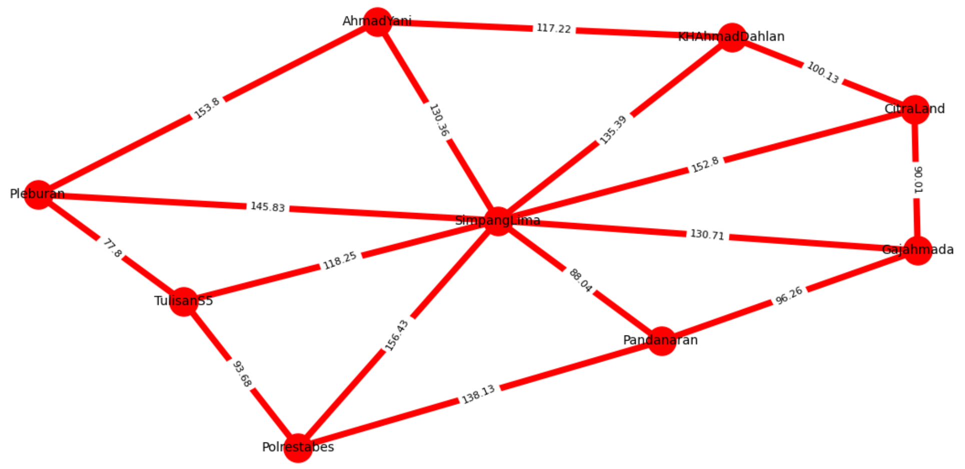
Contoh Shortest Path
Dari WarnetAradiw ke MetroIndahMall



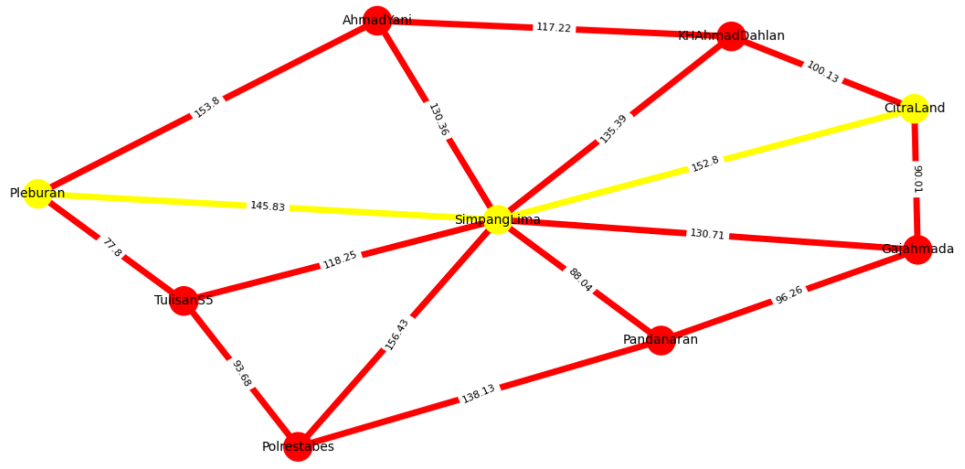
SimpangLima.txt


```
9
CitraLand -6.989097684573788 110.4235807573675
KHAhmadDahlan -6.989627554132646 110.42398730910085
AhmadYani -6.990653219240313 110.42407532599083
Pleburan -6.991636013978071 110.42310262054787
TulisanS5 -6.991213582345616 110.42250039715367
Polrestabes -6.9908827956368595 110.42168082532194
Pandanaran -6.989831323773677 110.42227798571564
Gajahmada -6.98908807583168 110.42278771965792
SimpangLima -6.990303688559784 110.42295687915843
0 100.13 0 0 0 0 0 90.01 152.80
100.13 0 117.22 0 0 0 0 0 135.39
0 117.22 0 153.8 0 0 0 0 130.36
0 0 153.8 0 77.8 0 0 0 145.83
0 0 0 77.8 0 93.68 0 0 118.25
0 0 0 0 93.68 0 138.13 0 156.43
0 0 0 0 0 138.13 0 96.26 88.04
90.01 0 0 0 0 0 96.26 0 130.71
152.80 135.39 130.36 145.83 118.25 156.43 88.04 130.71 0
```

Visualisasi Graph



Contoh Shortest Path
Dari CitraLand ke Pleburan



Link Program

https://github.com/fakhrinail/TucilStima3

Tabel Kelengkapan

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	