Tugas Kecil III IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Disusun oleh:

Karlsen Adiyasa Bachtiar (13519001)

Yudi Alfayat (13519051)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

Kode Program

1. **app.py**

```
from tkinter import filedialog
from tkinter import ttk
from tkinter import *

from Graph import *
from File import *
from Astar import findPath

SCREEN_WIDTH = 720
SCREEN_HEIGHT = 720
SCREEN_SIZE = f"{SCREEN_WIDTH}x{SCREEN_HEIGHT}"

graph = Graph(1, 0, [], [], [])

# Graph visualization

def showGraphVisualization(graph, path):
    if (graph.getNumOfNode() == 0):
        print("Graph is empty!!")
        return

    edges = []
    if (len(path) > 1):
        for i in range(0, len(path)-1):
            edge = []
            edge.append(path[i])
            edge.append(path[i+1])
            edges.append(edge)

    for i in range(graph.getNumOfNode()):
        friend = graph.getListConnectedNode(i)
        for j in range(len(friend)):
            idx = friend[j]
            if (idx > i):
                node1 = graph.getNode(i)
                node2 = graph.getNode(idx)
                distance = graph.getDistance(i, idx)
                xPos1 = getPosXRelative(graph, node1.x)
                yPos1 = getPosYRelative(graph, node1.y)

                xPos2 = getPosXRelative(graph, node2.x)
                yPos2 = getPosYRelative(graph, node2.y)
```

```
            color = "red"

            if (len(edges) > 0):
                e1 = [i, idx]
                e2 = [idx, i]
                if (e1 in edges or e2 in edges):
                    color = "black"


            drawLine(distance, xPos1, yPos1, xPos2, yPos2, color)

    for i in range(graph.getNumOfNode()):
        node = graph.getNode(i)

        xPos = getPosXRelative(graph, node.x)
        yPos = getPosYRelative(graph, node.y)

        color = "cyan"

        if (len(path) > 0):
            if (i in path):
                color = "orange"

        createNode(node.name, xPos, yPos, color)

def getPosXRelative(graph, x):
    pad = 70

    xPanelCenter = (graphVisualPanel.winfo_reqwidth()-4-pad)/2
    xCenterPos = (graph.getMaxX() + graph.getMinX())/2
    xMaxDisCenter = graph.getMaxDistanceX()/2

    dis = x - xCenterPos

    if (xMaxDisCenter == 0):
        return xPanelCenter + pad/2
    else:
        return dis/xMaxDisCenter * xPanelCenter + xPanelCenter + pad/2

def getPosYRelative(graph, y):
    pad = 70

    yPanelCenter = (graphVisualPanel.winfo_reqheight()-4-pad)/2
    yCenterPos = (graph.getMaxY() + graph.getMinY())/2
```

```
    yMaxDisCenter = graph.getMaxDistanceY()/2

    dis = y - yCenterPos

    if (yMaxDisCenter == 0):
        return yPanelCenter + pad/2
    else:
        return dis/yMaxDisCenter * yPanelCenter + yPanelCenter + pad/2


def createNode(name, xPos, yPos, color):
    radius = 20
    graphVisualPanel.create_oval(xPos-radius, yPos-radius, xPos+radius, yPos+radius, fill=color)
    graphVisualPanel.create_text(xPos, yPos, fill="darkblue", text=name)

def drawLine(dis, xPos1, yPos1, xPos2, yPos2, color):
    graphVisualPanel.create_line(xPos1, yPos1, xPos2, yPos2, fill=color, width=2)
    xTextPos = (xPos2 + xPos1)/2
    yTextPos = (yPos2 + yPos1)/2
    graphVisualPanel.create_text(xTextPos, yTextPos, text=round(dis, 2), fill="black")


def showMinimumPath(graph, path):
    print("Show minimum path")
    s = ""
    if (len(path) > 0):
        resetGraphVisualizationPanel()
        resetPathText()
        s = getStringPath(path)
        dis = graph.getDistancePath(path)
        sDis = "Distance : " + str(round(dis, 2)) + "\n"
        s = sDis + s
        pathText.insert(END, s)
        showGraphVisualization(graph, path)
    else:
        s = "Path is not Found!!"
        resetPathText()
        pathText.insert(END, s)

def showDistanceHeuristic(graph, heu):
    resetHeuText()
    row = 1
    val = ""
    for i in range(len(heu)):
        temp = ""
```

```
        name = graph.getNode(i).name
        temp += name
        temp += " : " + str(round(heu[i], 2)) + "; "
        if (len(val + temp) >= 50*row):
            row += 1
            val += '\n' + temp
        else:
            val += temp
    heuText.insert(END, val)

# End of Graph Visualization

def searchPath():
    print("A Star")
    global graph

    if (graph.getNumOfNode() > 0):

        if (nodeFromDropdown.current() < 0 or nodeToDropdown.current() < 0):
            print("Node is not selected!!")
        else:
            pathAndHeu = findPath(graph, nodeFromDropdown.current(), nodeToDropdown.current())
            path = pathAndHeu[0]
            showDistanceHeuristic(graph, pathAndHeu[1])
            showMinimumPath(graph, path)
    else:
        print("Graph is not ready!!")

def browse():
    print("browse!!")
    filePath = filedialog.askopenfilename(initialdir="/", title="Select Graph File", filetypes=(('text files', 'txt'),))
    # print(filePath)
    if (len(filePath) > 0):
        filePathText.delete(0, END)
        filePathText.insert(0, filePath)
        resetGraphVisualizationPanel()
        resetDropdown()
        resetHeuText()
        resetPathText()
        global graph
        graph = convertTextToGraph(filePath)

        if (graph.getNumOfNode() > 0):
            showGraphVisualization(graph, [])
            setDrowdownMenu(graph.getListNode())
```

```
def getStringPath(path):
    s = ""
    row = 1
    for i in range(len(path)):
        temp = ""
        name = graph.getNode(path[i]).name
        temp += name
        if (i != len(path)-1):
            temp += " -> "

        if (len(s+temp) >= 50*row):
            row += 1
            temp += '\n'

        s += temp

    return s

def resetGraphVisualizationPanel():
    graphVisualPanel.delete("all")

def resetPathText():
    pathText.delete("1.0", "end")

def resetHeuText():
    heuText.delete("1.0", "end")

def resetDropdown():
    print("Reset Dropdown")
    nodeFromDropdown.set('')
    nodeToDropdown.set('')
    nodeFromDropdown["values"] = []
    nodeToDropdown["values"] = []
    nodeFromDropdown.select_clear()
    nodeToDropdown.select_clear()

def setDrowdownMenu(nodes):
    val = []
    for i in range(len(nodes)):
        val.append(nodes[i].name)

    nodeFromDropdown["values"] = val
    nodeToDropdown["values"] = val
```

```
app = Tk()
app.title("A Start Path Planning")

frameLeft = Frame(app)
frameLeft.grid(row=0, column=0)
frameRight = Frame(app)
frameRight.grid(row=0, column=1)


# ========================================================
frame1 = Frame(frameLeft)
frame1.grid(row=0, padx=10, pady=10)

browseButton = Button(frame1, text="Browse", command=browse)
browseButton.grid(row=0, column=0)

filePathText = Entry(frame1, width=65)
filePathText.grid(row=0, column=1, columnspan=3, padx=10)

# ========================================================
frame2 = Frame(frameLeft)
frame2.grid(row=1, padx=10, pady=10)

labelTextFrom = Label(frame2, text="From : ")
labelTextFrom.grid(row=0, column=0)
nodeFromDropdown = ttk.Combobox(frame2, values=(), state="readonly")
nodeFromDropdown.grid(row=0, column=1)

fillLabel = Label(frame2, text=" ")
fillLabel.grid(row=0, column=2, padx=0)

labelTextTo = Label(frame2, text="To : ")
labelTextTo.grid(row=0, column=3)
nodeToDropdown = ttk.Combobox(frame2, values=(), state="readonly")
nodeToDropdown.grid(row=0, column=4)

searchButton = Button(frame2, text="Search", width=10, height=2, command=searchPath)
searchButton.grid(row=0, column=7, padx=20)

# ========================================================
frame3 = Frame(frameLeft)
frame3.grid(row=2, padx=10, pady=10)

pathTextLabel = Label(frame3, text="Path : ")
```

```
pathTextLabel.grid(row=0, column=0)
pathText = Text(frame3, height=10, width=50)
pathText.grid(row=1, column=0)

# ========================================================
frame4 = Frame(frameLeft)
frame4.grid(row=3, padx=10, pady=10)
heuTextLabel = Label(frame4, text="Heuristic : ")
heuTextLabel.grid(row=0, column=0)
heuText = Text(frame4, height=10, width=50)
heuText.grid(row=1, column=0)

# ========================================================
frame5 = Frame(frameRight, width=610, height=610)
frame5.grid(row=0, padx=10, pady=10)

graphVisualPanel = Canvas(frame5, width=700, height=700, bg="light grey")
graphVisualPanel.grid(row=0, pady=5, padx=5)

app.mainloop()
```

2. File.py

```
from Graph import *

def convertTextToGraph(filePath):

    try:
        if (len(filePath) < 4):
            dummy = Graph()
            return dummy

        dirNode = filePath
        dirAdj = filePath[:len(filePath)-4] + "_adj.txt"

        fileNode = open(dirNode, "r")
        fileAdj = open(dirAdj, "r")

        scale = int(fileNode.readline().split(",")[0])

        numOfNode = int(fileNode.readline().split(",")[0])

        graph = Graph(scale, 0, [], [], [])

        for line in fileNode:
```

```
            data = line.split(",")
            if (len(data) < 3+1):
                dummy = Graph()
                return dummy

            x = float(data[0])
            y = float(data[1])
            name = data[2]
            node = Node(name, x, y)

            graph.addNode(node)

        for i in range(numOfNode):
            line = fileAdj.readline()
            data = line.split(",")
            if (len(data) < numOfNode+1):
                dummy = Graph()
                return dummy

            for j in range(numOfNode):
                if (data[j] == '1'):
                    graph.addConnectedNode(i, j)

        return graph

    except IOError:
        print("Graph cannot be converted!!!")
        dummy = Graph()
        return dummy
```

3. Graph.py

```
class Node:
    def __init__(self, name, x, y):
        self.name = name
        self.x = x
        self.y = y

    def printNode(self):
        print(self.name + " : " + str(self.x) + ", " + str(self.y))


class Graph:

    # def __init__(self):
```

```
    # self.numOfNode = 0
    # self.nodes = []               # array of node
    # self.numOfConnectedNode = []    # array of integer
    # self.connectedNode = []         # array of array of integer

  def __init__(self, scaleTemp = 1, numOfNodeTemp = 0, nodesTemp = [], numOfConnectedNodeTemp = [],
connectedNodeTemp = []):
    self.scale = scaleTemp
    self.numOfNode = numOfNodeTemp
    self.nodes = nodesTemp
    self.numOfConnectedNode = numOfConnectedNodeTemp
    self.connectedNode = connectedNodeTemp

  def getListNode(self):
    return self.nodes

  def getNode(self, idxNode):
    return self.nodes[idxNode]

  def getListConnectedNode(self, idxNode):
    return self.connectedNode[idxNode]

  def getListListConnected(self):
    return self.connectedNode

  def getListNumOfConnectedNode(self):
    return self.numOfConnectedNode

  def getNumOfConnectedNode(self, idxNode):
    return self.numOfConnectedNode[idxNode]

  def getNumOfNode(self):
    return self.numOfNode

  def getIdxConnectedNode(self, idxNode, idx):
    return self.connectedNode[idxNode][idx]

  def getConnectedNode(self, idxNode, idxConnect):
    return self.nodes[self.getIdxConnectedNode(idxNode, idxConnect)]

  def getIdxNode(self, node):
    count = 0
    for x in self.nodes:
      if x.name == node:
        return count
```

```
        else:
            count = count + 1

    def addNode(self, newNode):
        self.numOfNode += 1
        self.nodes.append(newNode)
        self.numOfConnectedNode.append(0)
        self.connectedNode.append([])

    def addConnectedNode(self, idxNode, idxConnect):
        self.numOfConnectedNode[idxNode] += 1
        self.connectedNode[idxNode].append(idxConnect)

    def addEdge(self, idx1, idx2):
        self.addConnectedNode(idx1, idx2)
        self.addConnectedNode(idx2, idx1)

    def isExistEdge(self, idx1, idx2):
        for x in self.connectedNode[idx1]:
            if x == idx2:
                return True

        return False

    def getDistance(self, idx1, idx2):
        x1 = self.nodes[idx1].x
        y1 = self.nodes[idx1].y

        x2 = self.nodes[idx2].x
        y2 = self.nodes[idx2].y

        # Sementara pakau eucludian
        return (((x2-x1)**2 + (y2-y1)**2)**(1/2)) * self.scale


    def getMinX(self):
        if (self.numOfNode <= 0):
            return -999
        else:
            minX = self.nodes[0].x
            for i in range(1, self.numOfNode):
                if (minX > self.nodes[i].x):
                    minX = self.nodes[i].x

            return minX
```

```
def getMaxX(self):
    if (self.numOfNode <= 0):
        return -999
    else:
        maxX = self.nodes[0].x
        for i in range(1, self.numOfNode):
            if (maxX < self.nodes[i].x):
                maxX = self.nodes[i].x

        return maxX

def getMinY(self):
    if (self.numOfNode <= 0):
        return -999
    else:
        minY = self.nodes[0].y
        for i in range(1, self.numOfNode):
            if (minY > self.nodes[i].y):
                minY = self.nodes[i].y

        return minY

def getMaxY(self):
    if (self.numOfNode <= 0):
        return -999
    else:
        maxY = self.nodes[0].y
        for i in range(1, self.numOfNode):
            if (maxY < self.nodes[i].y):
                maxY = self.nodes[i].y

        return maxY

def getDistancePath(self, path):
    sum = 0
    for i in range(len(path)-1):
        idx1 = path[i]
        idx2 = path[i+1]
        sum += self.getDistance(idx1, idx2)

    return sum

def getMaxDistanceX(self):
    return self.getMaxX() - self.getMinX()
```

```
    def getMaxDistanceY(self):
        return self.getMaxY() - self.getMinY()

    def printGraph(self):
        for i in range(self.numOfNode):
            name = self.nodes[i].name
            x = self.nodes[i].x
            y = self.nodes[i].y
            numOfconnect = self.numOfConnectedNode[i]
            # print(numOfconnect)
            data = name + "[" + str(x) + "," + str(y) + "]" + str(numOfconnect)
            print(data, end=", ")
        print()

        for i in range(self.numOfNode):
            print(self.connectedNode[i])
```

4. Astar.py

```
from Graph import *

def getHeuristic(graph, destination):
    heu = []
    for i in range(graph.getNumOfNode()):
        dis = graph.getDistance(i, destination)
        heu.append(dis)

    return heu

def getMinDistance(dis):
    if (len(dis) <= 0):
        return -1
    else:
        idxMin = 0
        for i in range(1, len(dis)):
            tup = dis[i]
            tupMin = dis[idxMin]

            if (tup[1] < tupMin[1]):
                idxMin = i

        return idxMin

def astar(graph, idxDestinastion, heu, distance, stack, blacklist):
```

```
    if (stack[-1] == idxDestinastion):
        return stack

    idxFrom = stack[-1]

    friend = graph.getListConnectedNode(idxFrom)
    count = 0
    for i in range(len(friend)):
        idxFriend = friend[i]

        if (idxFriend not in stack):
            gn = graph.getDistance(idxFrom, idxFriend)
            hn = heu[idxFriend]
            fn = gn + hn
            tempStack = [e for e in stack]
            tempStack.append(idxFriend)

            tempDistance = [tempStack, fn]

            if (tempDistance not in distance and tempStack not in blacklist):
                distance.append(tempDistance)
                count += 1


    if (count == 0 and len(distance) != 0):
        idxMinDistance = getMinDistance(distance)
        blacklist.append(distance[idxMinDistance][0])
        distance.pop(idxMinDistance)

    if (len(distance) == 0):
        return []

    idxMinDistance = getMinDistance(distance)
    tup = distance[idxMinDistance]
    stack = tup[0]

    if (stack[-1] == idxDestinastion):
        return stack

    return astar(graph, idxDestinastion, heu, distance, stack, blacklist)



def findPath(graph, idxFrom, idxTo):
    if (graph.getNumOfNode() > 0):
```

```
        heu = getHeuristic(graph, idxTo)
        path = astar(graph, idxTo, heu, [], [idxFrom], [])

        if (len(path) <= 0):
            return [[], heu]
        else:
            return [path, heu]
    else:
        return [[], []]
```

Bonus

5. **map.py**

```
import folium
from folium import plugins
from Astar import *
from Graph import *
from File import *


def printPath(path):
    for x in path:
        print(x+1)


def inputRouteAstar(path, route_Astar):
    for idxNode in path:
        node = g1.nodes[idxNode]
        route_Astar.append([node.x, node.y])

def createMarkers(graph):
    # Create Markers
    i = 1
    heu = getHeuristic(graph, idxTo-1)
    for coor in graph.nodes:
        num = str(i)
        koordinat = str(coor.x) + " " + str(coor.y)
        folium.Marker([coor.x, coor.y],
                      popup=heu[i-1],
                      tooltip="Click for more info",
                      icon=plugins.BeautifyIcon(number=i,
                                                border_color='blue',
```

```
                                    border_width=2,
                                    text_color='red',
                                    inner_icon_style='margin-top:0px;')).add_to(map1)
        i = i + 1


# Create map object
g1 = convertTextToGraph("../test/ITB.txt")

map1 = folium.Map(location=[-6.892650, 107.610433], zoom_start=20)


numOfNode = g1.getNumOfNode()

print("Masukkan satu angka diantara 1 sampai ", numOfNode)
idxFrom = int(input("Masukkan simpul mulai (dalam integer) : "))
idxTo = int(input("Masukkan simpul tujuan (dalam integer) : "))

while idxFrom < 1 or idxFrom > numOfNode or idxTo < 1 or idxTo > numOfNode:
    print("\nMasukkan tidak valid. Coba lagi\n")
    print("Masukkan satu angka diantara 1 sampai ", numOfNode)
    idxFrom = int(input("Masukkan simpul mulai (dalam integer) : "))
    idxTo = int(input("Masukkan simpul tujuan (dalam integer) : "))


createMarkers(g1)

route_Graph = [
    [-6.892650, 107.610433],  # 1
    [-6.892650, 107.608763],  # 2
    [-6.891056, 107.608712],  # 3
    [-6.891057, 107.609713],  # 4
    [-6.891037, 107.611024],  # 6
    [-6.891057, 107.609713],  # 4
    [-6.891934, 107.610388],  # 5
    [-6.891362, 107.611052],  # 8
    [-6.891037, 107.611024],  # 6
    [-6.890978, 107.612087],  # 7
    [-6.891355, 107.612193],  # 9
    [-6.891362, 107.611052],  # 8
    [-6.891355, 107.612193],  # 9
    [-6.892427, 107.612027],  # 10
    [-6.892650, 107.610433],  # 1
    [-6.891934, 107.610388]   # 5
]
```

```
pathAndHeu = findPath(g1, idxFrom-1, idxTo-1)
path = pathAndHeu[0]
printPath(path)


route_Astar = []
inputRouteAstar(path, route_Astar)


# add route to map
folium.PolyLine(route_Graph).add_to(map1)

# add ant path route to map
plugins.AntPath(route_Astar).add_to(map1)

# Generate map
map1.save('map.html')
```

Peta dan Graf input

01.txt

1,

8,

5,5,A,

20,7,B,

15,20,C,

5,18,D,

25,15,E,

25,25,F,

7,23,G,

30,21,H,

01_adj.txt

0,1,1,1,0,0,0,0,

1,0,1,0,1,1,0,0,

1,1,0,0,0,1,1,0,

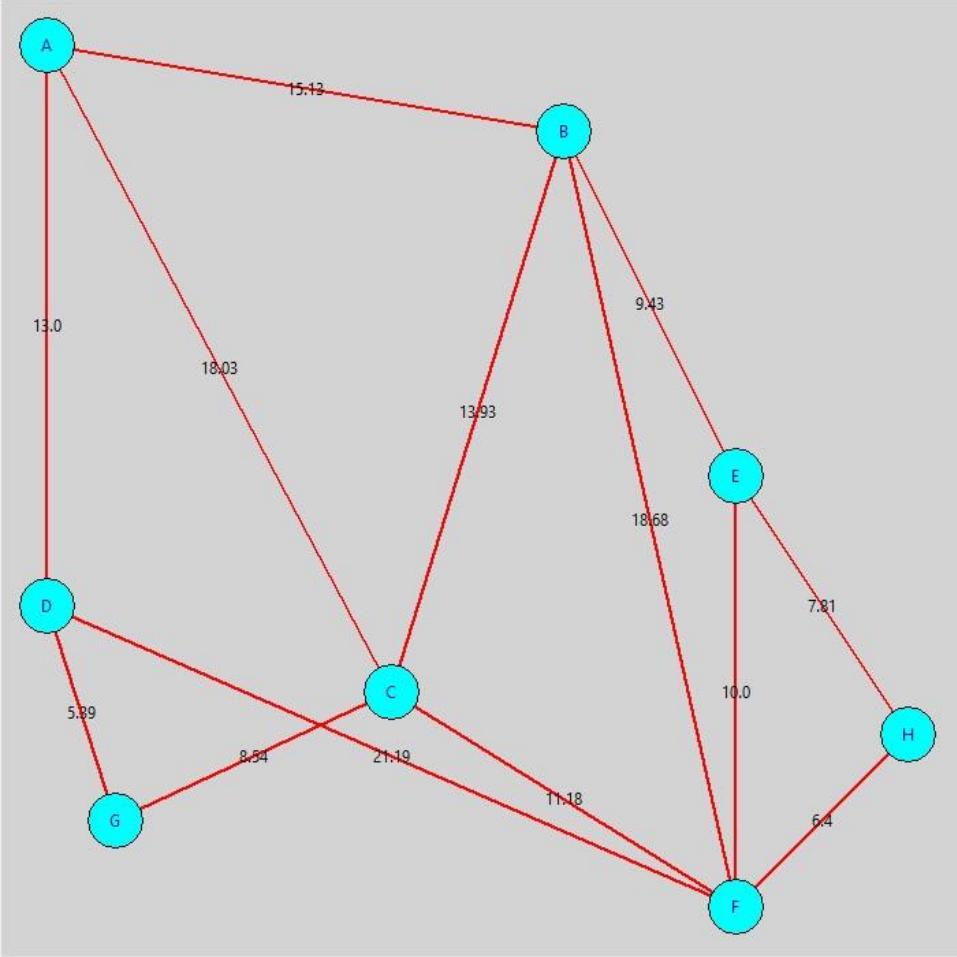
1,0,0,0,0,1,1,0,

0,1,0,0,0,1,0,1,

0,1,1,1,1,0,0,1,

0,0,1,1,0,0,0,0,

0,0,0,0,1,1,0,0,



02.txt

1,

8,

5,5,A,

20,7,B,

15,20,C,

5,18,D,

25,15,E,

25,25,F,

7,23,G,

30,21,H,

02_adj.txt

0,1,1,1,0,0,0,0,

1,0,1,0,1,1,0,0,

1,1,0,0,0,1,1,0,

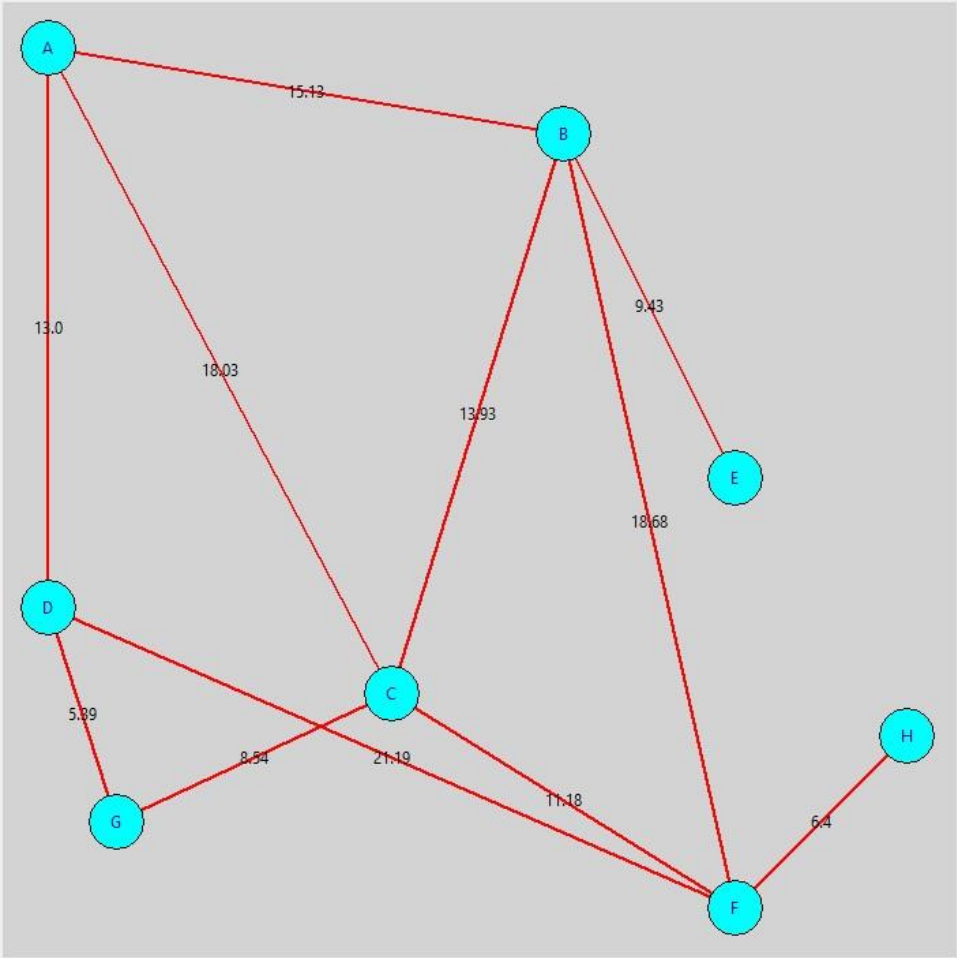
1,0,0,0,0,1,1,0,

0,1,0,0,0,0,0,0,

0,1,1,1,0,0,0,1,

0,0,1,1,0,0,0,0,

0,0,0,0,0,1,0,0,



03.txt

1,

8,

5,5,A,

20,7,B,

15,20,C,

5,18,D,

25,15,E,

25,25,F,

7,23,G,

30,21,H,

03_adj.txt

0,1,1,1,0,0,0,0,

1,0,1,0,1,1,0,0,

1,1,0,0,0,1,1,0,

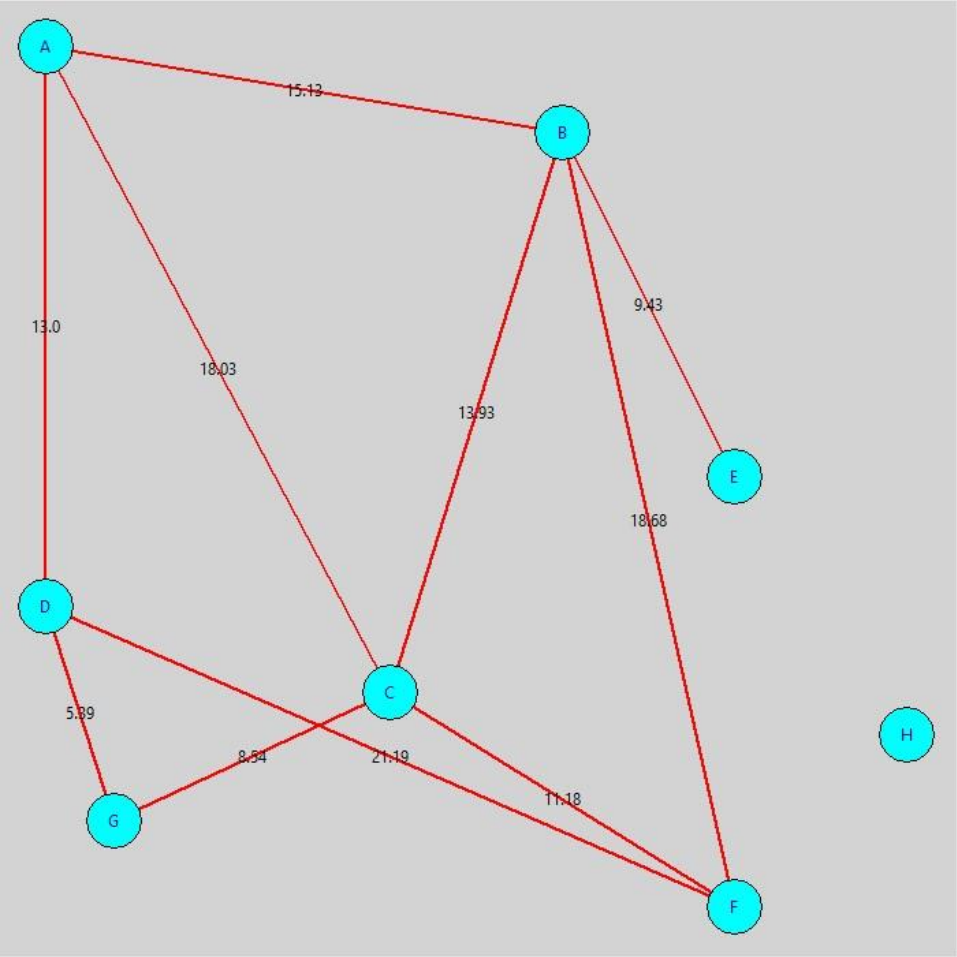
1,0,0,0,0,1,1,0,

0,1,0,0,0,0,0,0,

0,1,1,1,0,0,0,0,

0,0,1,1,0,0,0,0,

0,0,0,0,0,0,0,0,



ITB.txt merupakan peta ITB jalan ganesha

ITB.txt

100000,

10,

-6.892650, 107.610433,A,

-6.892650, 107.608763,B,

-6.891056, 107.608712,C,

-6.891057, 107.609713,D,

-6.891934, 107.610388,E,

-6.891037, 107.611024,F,

-6.890978, 107.612087,G,

-6.891362, 107.611052,H,

-6.891355, 107.612193,I,

-6.892427, 107.612027,J,

ITB_adj.txt

0,1,0,0,1,0,0,0,0,0,

1,0,1,0,0,0,0,0,0,0,

0,1,0,1,0,0,0,0,0,0,

0,0,1,0,1,0,0,0,0,0,

1,0,0,1,0,0,0,1,0,0,

0,0,0,0,0,0,0,1,0,0,

0,0,0,0,0,0,0,1,0,0,

0,0,0,0,1,1,1,0,1,0,

0,0,0,0,0,0,0,1,0,1,

0,0,0,0,0,0,0,0,1,0,

AlunAlun.txt

100000,

10,

-6.921233, 107.607760,A,

-6.921036, 107.606462,B,

-6.920924, 107.605099,C,

-6.920855, 107.604080,D,

-6.922075, 107.604022,E,

-6.923119, 107.603942,F,

-6.923417, 107.606222,G,

-6.922405, 107.606410,H,

-6.922565, 107.607628,I,

-6.922687, 107.608433,J,

AlunAlun_adj.txt

0,1,0,0,0,0,0,0,1,0,

1,0,1,0,0,0,0,0,0,0,

0,1,0,1,0,0,0,0,0,0,

0,0,1,0,1,0,0,0,0,0,

0,0,0,1,0,1,0,1,0,0,

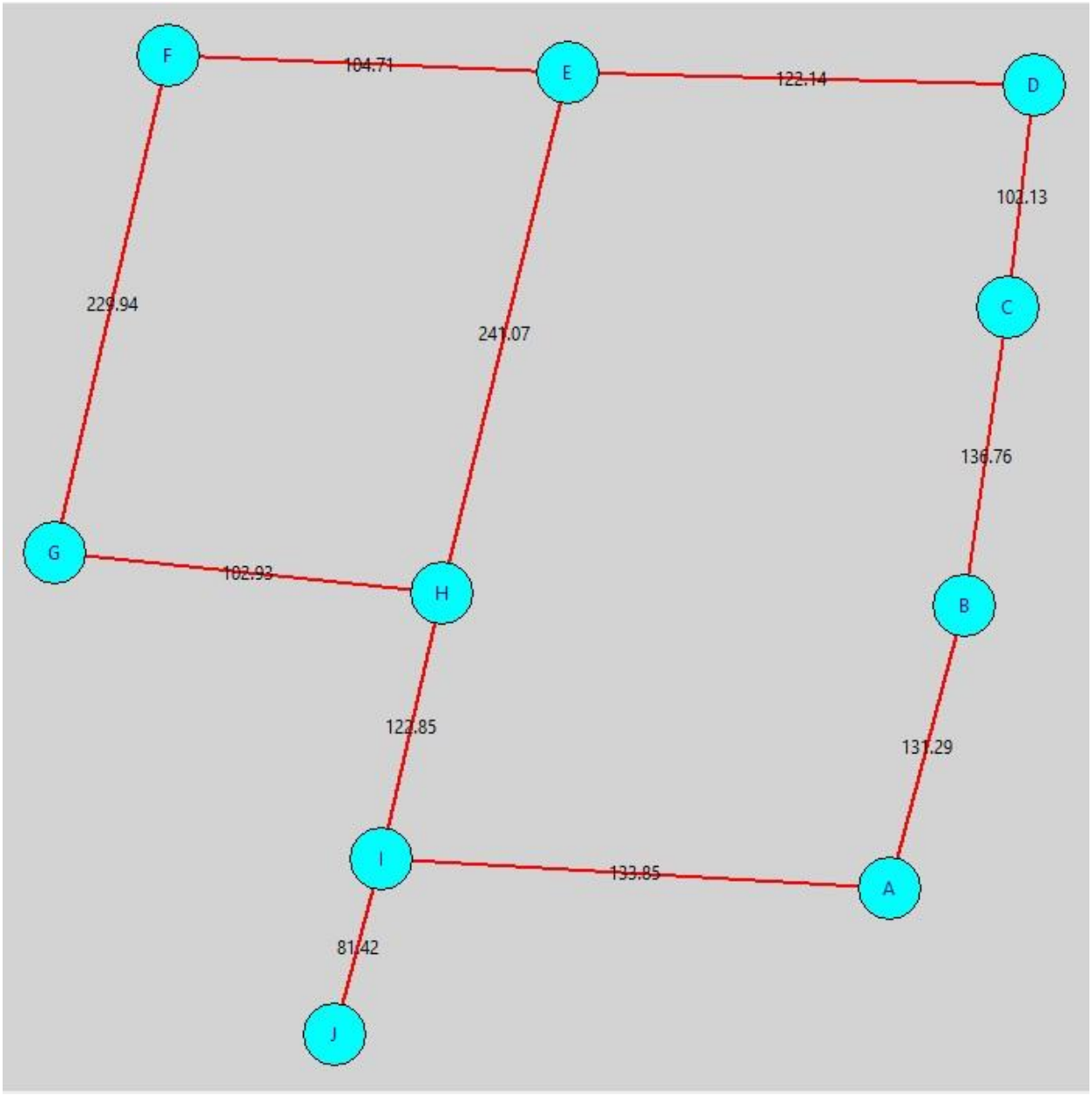
0,0,0,0,1,0,1,0,0,0,

0,0,0,0,0,1,0,1,0,0,

0,0,0,0,1,0,1,0,1,0,

1,0,0,0,0,0,0,1,0,1,

0,0,0,0,0,0,0,0,1,0,


F
104.71
E
122.14
D
102.13
C
229.94
241.07
136.76
G
102.93
H
B
122.85
131.29
I
133.85
A
81.42
J

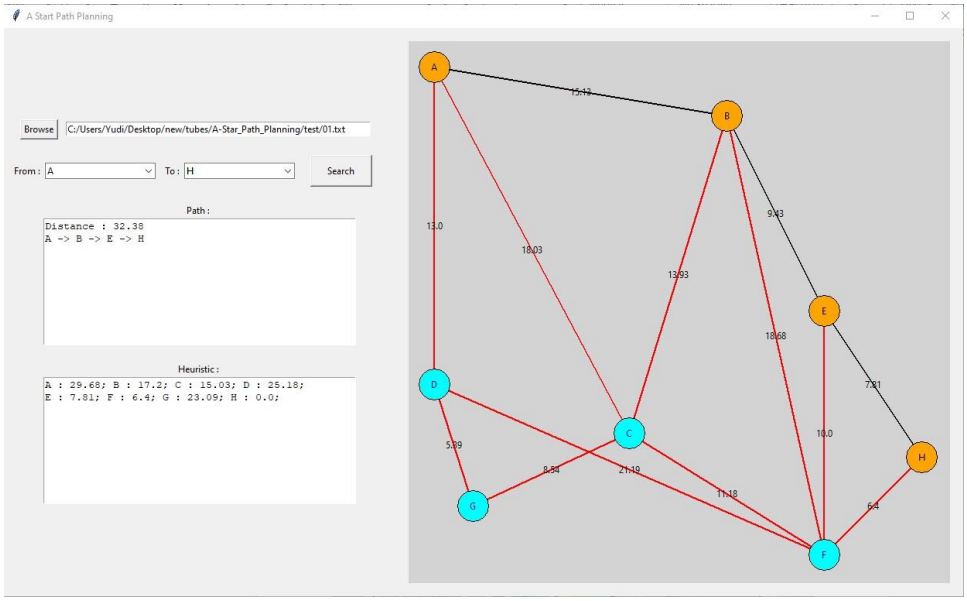
Screenshot

Wajib

1. 01.txt

   Node Awal : A

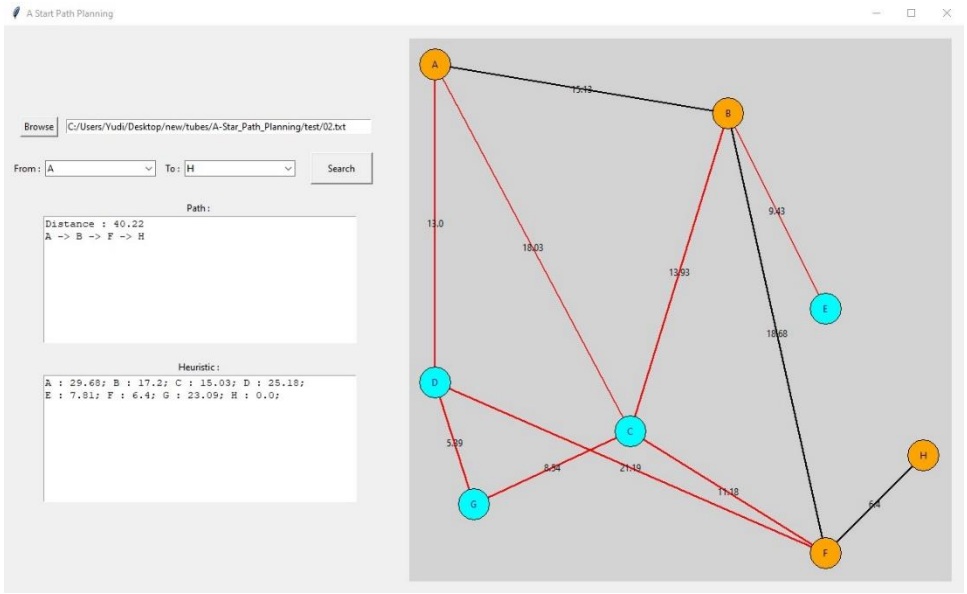
   Node Tujuan : H



2. 02.txt

   Node Awal : A

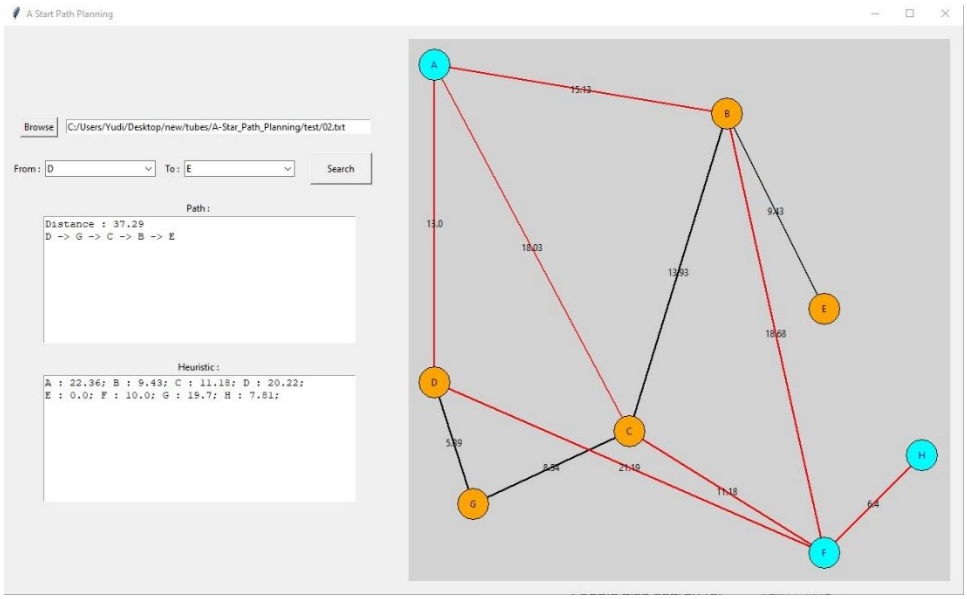
   Node Tujuan : H



3. 02.txt

   Node Awal : D

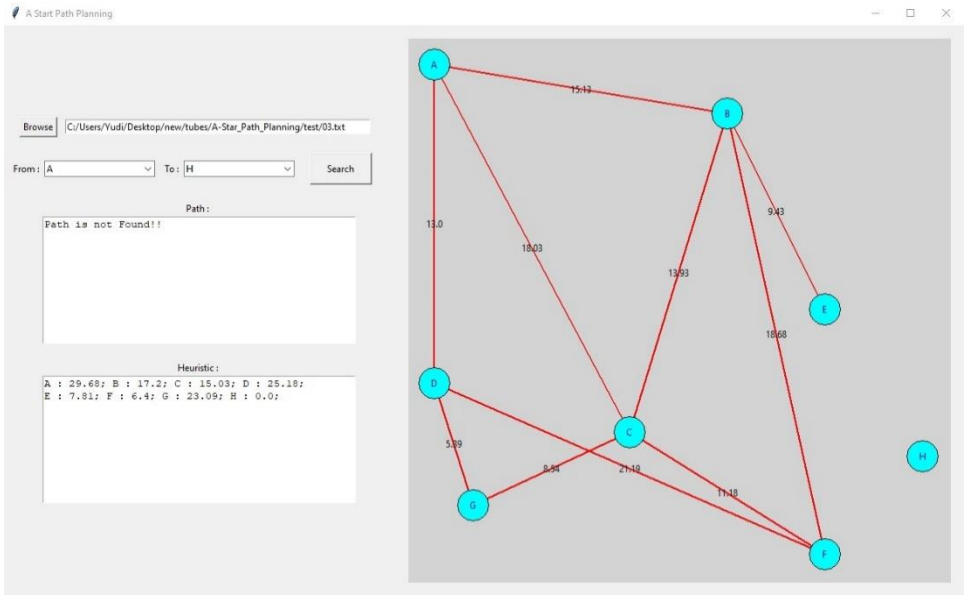
   Node Tujuan : E



4. 03.txt

   Node Awal : A

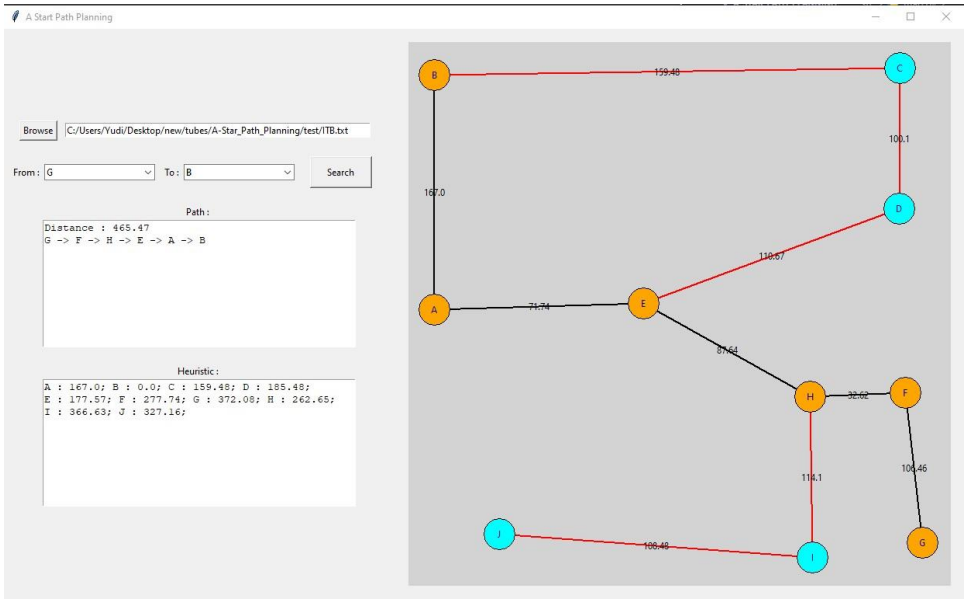
   Node Tujuan : H



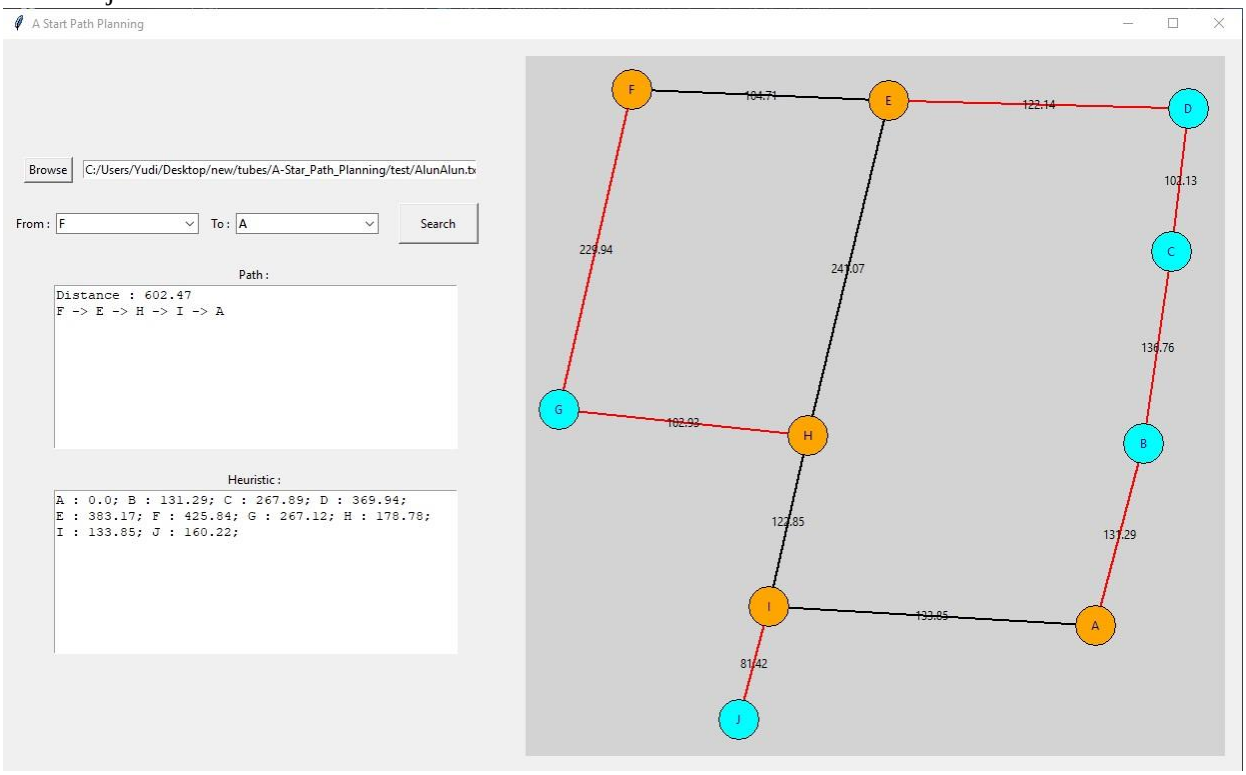
5. ITB.txt

   Node Awal : G

   Node Tujuan : B



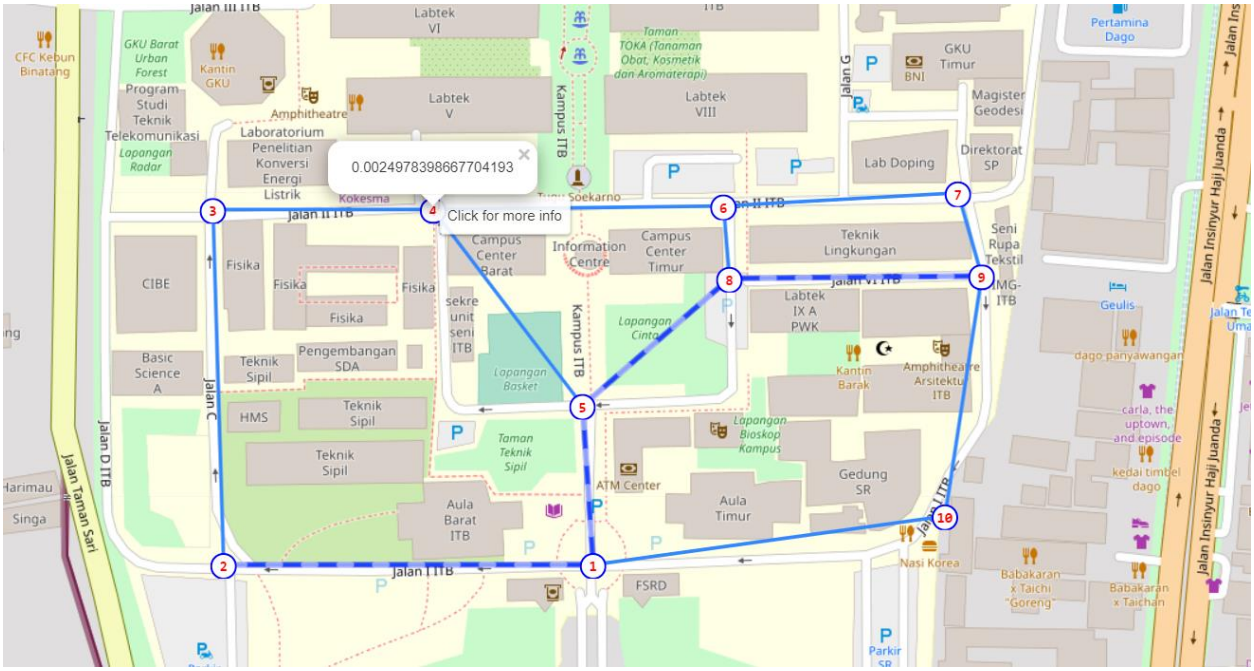
6. AlunAlun.txt
   Node Awal : F
   Node Tujuan : A



Bonus

ITB.txt

Setiap node jika diklik akan menampilkan nilai heuristik terhadap node tujuan

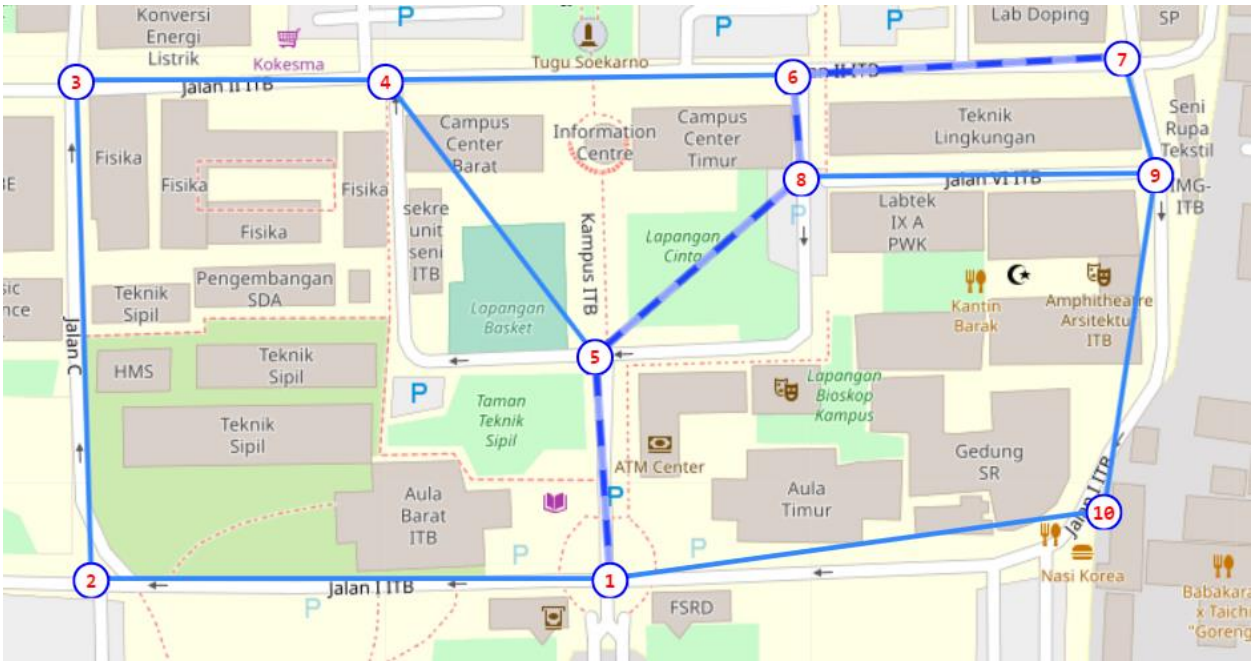


Node awal =2

Node tujuan = 9

Node awal =2

Node tujuan = 7



Node awal = 7

Node tujuan = 1

AlunAlun.txt

Node Awal : 1

Node Akhir : 6

Link github

https://github.com/yudialfayat/A-Star_Path_Planning.git

Centang (V) jika ya

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