LAPORAN TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

IMPLEMENTASI ALGORITMA A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Oleh:

Hizkia Raditya Pratama Roosadi	**K02/13519087**
Richard Rivaldo	**K04/13519185**

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

1. Kode Program

Alur kerja program ini adalah sebagai berikut. Pertama, pengguna harus memasukkan sebuah *file* yang contoh format atau susunan filenya bisa dilihat pada contoh *testcase* di bagian berikutnya. *File* ini bisa diletakkan pada direktori yang sama dengan *source code* program yang akan dieksekusi untuk mempermudah penginputan nama sehingga tidak membutuhkan direktori lengkap pada saat input tersebut.

Kemudian, program akan membaca berkas tersebut dan menjadikannya dalam bentuk representasi kelas graf. Setelah itu, program akan menampilkan visualisasi graf secara penuh melalui NetworkX. Untuk melanjutkan program, maka pengguna harus menutup *windows* atau tampilan NetworkX terlebih dahulu.

Pengguna kemudian diberikan pilihan untuk melanjutkan proses pencarian A*, menginput *file* masukan lagi, atau keluar dari program. Pilihan ini diberikan selama pengguna belum memberhentikan program. Jika pengguna memilih untuk melakukan pencarian A*, maka kemudian program akan menampilkan daftar nama simpul yang ada di dalam graf, serta meminta input yang diperlukan untuk A* tersebut, yaitu nama simpul asal dan simpul tujuan.

Program akan mengeluarkan keluaran yang sesuai dengan input yang dimasukkan pengguna dan algoritma A*. Jika ditemukan jalur terpendek untuk kedua simpul, maka program akan memberikan pilihan bagi pengguna untuk memilih metode visualisasi yang diinginkan. Lagi-lagi, pengguna harus menutup *windows* untuk visualisasi dengan NetworkX, atau menyelesaikan koneksi dengan server lokal jika divisualisasi dengan HERE dan Flask, untuk melanjutkan program.

Jika pengguna memilih untuk memasukkan input file baru, maka program akan meminta input dari pengguna lagi dan akan menampilkan visualisasi yang sesuai. Jika pengguna memilih untuk keluar dari program, maka program akan berhenti. Proses ini akan terus berlangsung sampai pengguna memutuskan untuk keluar dari program.

Implementasi algoritma A* pada program ini menggunakan 2 nilai yaitu Haversine distance (jarak antara dua buah koordinat lintang, bujur pada permukaan bumi) dan value sisi pada graf (yang diturunkan juga dari Haversine distance). Selain itu, program juga akan menerima 2 Node yang akan menjadi Node awal dan tujuan. Program akan membaca graf dan mengekspansi node dengan cara dimasukkan kedalam sebuah priority queue yang dapat menyimpan prioritas dari setiap Node. Prioritas dihitung dari Haversine distance + value sisi. Selain itu, program juga mengimplementasikan parent yang menunjuk pada Node yang membangkitkan Node baru hasil ekspansi. Terakhir, program akan melakukan backtracking dari Node tujuan melalui jalur parent hingga sampai ke Node awal.

Node.py

```
from collections import defaultdict
from math import *
```

```
# Node class
class Node:
    def __init__(self, name, x, y):
        # User-Defined Constructor, no need default
constructor
        self.name = name
        self.x = x
        self.y = y
        self.neighbors = defaultdict(lambda: "No
neighbors")
        self.parent = []

    def getDistanceBetween(self, otherNode):
        # Euclidean Distance
        x = self.x - otherNode.x
        y = self.y - otherNode.y

        return sqrt(x ** 2 + y ** 2)

    def calculateHaversine(self, otherNode):
        # Earth Radius, Get Haversine in KM
        earthRadius = 6371

        # Convert Longitude and Latitude to Radians
        lat1 = radians(self.x)
        long1 = radians(self.y)
        lat2 = radians(otherNode.x)
        long2 = radians(otherNode.y)
        # Get the difference
        latDiff = lat2 - lat1
        longDiff = long2 - long1
        # Haversine
        a = (sin(latDiff / 2)**2) + (cos(lat1) * cos(lat2)
* sin(longDiff / 2)**2)
        c = 2 * asin(sqrt(a))

        return(c* earthRadius)

    def printNode(self):
        # Get information from the node
        print("%s (%d, %d)" % (self.name, self.x, self.y))
        print("List of neighbors:")
        for key, value in self.neighbors.items():
            print(key.name, ":", value)
```

```
    def containsNeighbor(self,node):
        # Check if a node contains other node as neighbors
        for key,value in self.neighbors:
            if (node.name==key.name):
                return True
        return False

    def addParent(self,parentNode):
        # Add the parent of the node
        self.parent.append(parentNode)

    def sortNeighbors(self):
        # Sort neighbors based on Haversine Distance
        self.neighbors = {k: v for k, v in
sorted(self.neighbors.items(), key=lambda item: item[1])}

    def getParents(self):
        # Get the parents of a node
        if (len(self.parent)!=0):
            return self.parent[0]
        else:
            return []
    def hasParents(self):
        # Check if a node has any parent
        return (len(self.parent)!=0)

    def removeAllParent(self):
        #remove all parent from node
        self.parent.clear()

    def sameName(self,Node):
        # Check uf two node has same name
        return (self.name==Node.name)
```

Graph.py

```
# Libraries
import matplotlib.pyplot as plt
import networkx as nx
from Utils import *
from Node import *

# Graph Class
```

```
class Graph:
    def __init__(self, filename):
        self.nodeList = []

        # Number of nodes from the source
        numOfNodes = getNodeNumber(filename)
        #will result in 3 arrays of the same size each
containing
        #name,x,and y, that will be used for nodes
        nameArray = getArrayFromFile(0, numOfNodes,
filename)
        xArray = getArrayFromFile(1, numOfNodes, filename)
        yArray = getArrayFromFile(2, numOfNodes, filename)

        # Append the nodes to nodeList
        for i in range (0, (numOfNodes)):
            self.nodeList.append(Node(nameArray[i],
float(xArray[i]), float(yArray[i])))
        # Set neighbors
        self.setNeighbors(filename)

    def setNeighbors(self, filename):
        # Add Neighbors from the list of nodes to the list
of neighbors
        adjMatrix = getAdjMatrix(filename)

        # Get the corresponding row
        for i in range(len(self.nodeList)):
            dictOfNeighbors = defaultdict(lambda: "No
Nodes")
            weightRow = adjMatrix[i]
            for j in range(len(weightRow)):
            # Get the neighbors
                if(weightRow[j] == 1):
                    # Append K: Name of neighbors, V: Edge
Weight
                    weight =
self.nodeList[i].calculateHaversine(self.nodeList[j])
                    dictOfNeighbors[self.nodeList[j]] =
weight
            self.nodeList[i].neighbors = dictOfNeighbors

    def searchByName(self, name):
        # Search a  node based on its name
        for node in self.nodeList:
            if(node.name == name):
```

```
                return node

    def checkGraph(self):
        # Iterate Node List
        for i in range (0, len(self.nodeList)):
            print("Node %d: " % (i + 1))
            self.nodeList[i].printNode()
            print()

    def visualize(self, pathList):
        # Create NX Graph
        graph = nx.Graph()
        path = toAdjacencyList(pathList)

        # Iterate each node
        for node in self.nodeList:
            # Add nodes with node names
            graph.add_node(node.name)

            # Add edges if not exists in the edge list of
NX Graph
            for key, value in node.neighbors.items():
                # If the path list is not empty
                # Check if the nodes are in the
corresponding edges
                # Give different colors for both case,
highlight solution path
                if(checkIfInEdges(node.name, key.name,
path)):
                    graph.add_edge(node.name, key.name,
color = 'r', weight = round(value, 2))
                elif(not graph.has_edge(node.name,
key.name)):
                    graph.add_edge(node.name, key.name,
color = 'b', weight = round(value, 2))
        # Create a layout
        pos=nx.spring_layout(graph)
        # Draw graph nodes
        edges,colors = zip(*nx.get_edge_attributes(graph,
'color').items())
        nx.draw(graph, pos, edgelist=edges,
edge_color=colors, with_labels = True, font_weight =
'bold')
        edge_weight = nx.get_edge_attributes(graph,
'weight') # Get graph edges weights
        # Draw graph edges
```

```
        nx.draw_networkx_edge_labels(graph, pos,
edge_labels = edge_weight)
        print("Successfully visualize the graph. Close the
NetworkX Visualization to continue!")
        plt.show()

    def isNodeInGraphByName(self,nodeName):
        # Find if a Node is in the graph by its name,
returns true if yes
        for n in self.nodeList:
            if (n.name==nodeName):
                return True
        return False

    def searchNodeByNode(self,Node):
        # Find a Node by its name
        for n in self.nodeList:
            if (n.name==Node.name):
                return n

    def getPoints(self):
        # Get each nodes informations
        points = []
        for node in self.nodeList:
            nodePoint = []
            nodePoint.append(node.x)
            nodePoint.append(node.y)
            nodePoint.append(node.name)
            points.append(nodePoint)
        return points

    def showAllPlaces(self):
        # Show all nodes name
        print("List of available places: ")
        for node in self.nodeList:
            print(node.name)
        print()

    def removeAllNodeParent(self):
        # Reset all parents from all nodes in the graph
        for item in self.nodeList:
            item.removeAllParent()
```

Utils.py

```
def getNodeNumber(filename):
    # Getting the number of nodes
    source = open(filename)
    initNumber = [0]; # get number of nodes first

    for position, line in enumerate(source):
        if position in initNumber:
            numOfNodes = int(line) # acquire num of nodes
    return numOfNodes

def getArrayFromFile(column, number, filename):
    #returns array from filename of a specific column
    source = open(filename)
    arr = []

    for i, line in enumerate(source):
        if i >= 1 and i <= number:
            arr.append((line.replace('\n','')).split('
')[column])
    return arr

def getAdjMatrix(filename):
    # Make an Adjacency Matrix
    adjMatrix = []

    # Open and get number of nodes from the file
    source = open(filename)
    numOfNodes = getNodeNumber(filename)

    # Enumerate the files and iterate
    for position, line in enumerate(source):
        line = line.replace('\n','').split(' ') # Clean the
file
        row = []
        if(position >= numOfNodes + 1):
            for weight in line:
                row.append(int(weight)) # Get each row of
the file containing weights
            adjMatrix.append(row) # Append to the matrix

    return adjMatrix

def containsNode(arr,Node):
    # Procedure to check if an array contains a node
```

```
    for item in arr:
        if (item.name==Node.name and item.x==Node.x and
item.y==Node.y):
            return True
    return False

def isNeighbors(node1,node2):
    # Returns true if node1 and node2 are neighbors
    for key,value in node1.neighbors.items():
        if (key.name==node2.name and key.x==node2.x and
key.y==node2.y):
            return True
    return False

def getMinimumAStarNode(myGraph, myNode,
endNode,currentValue):
    # A Star implementation is here (hopefully, can change)
    # Returns node with the minimum distance to goal from
myNode
    curMinNode = myNode; #set current minimum node as self
    count = 999999; #set count
    for key,value in myNode.neighbors.items():
        currentNode = myGraph.searchNodeByNode(key)
        curCount = value +
currentNode.calculateHaversine(endNode)+currentValue;
#check value + distance between nodes
        if (curCount<count): #if currentCount<count make
currentMinimumNode = currentNode
            count = curCount
            curMinNode = currentNode

    return curMinNode

def is_in_queue(x,q):
    # Check if a node is in the queue
    for items in q.queue:
        if (items[1].name==x.name):
            return True
    return False

def findInQueue(x,q):
    # Find a node in the queue
    for items in q.queue:
        if(items[1].name==x.name):
            return items[1]
    return None
```

```
def toAdjacencyList(pathList):
    # Change path to list of edges
    edges = []
    for i in range(len(pathList)):
        if(i != (len(pathList) - 1)):
            edge = []
            edge.append(pathList[i].name)
            edge.append(pathList[i + 1].name)
            edges.append(edge)
    return edges

def checkIfInEdges(node1, node2, edges):
    # Check if two nodes are in the edges list
    for i in range(len(edges)):
        if(edges[i][0] == node1 and edges[i][1] == node2 or
           edges[i][0] == node2 and edges[i][1] == node1):
            return True
    return False
```

Tucil3.py

```
#Tucil 3 Stima
#Hizkia R. 13519087
#Richard R. 13519185

# Import Libraries
from flask import Flask, render_template
from queue import PriorityQueue
from Graph import *

def printqueue(q):
    for items in q.queue:
        if (not items[1].hasParents()):
            print("No items in the queue!")
        else:
            print (str(items[0])+ " node " + items[1].name+
" with parent " + items[1].getParents().name)
# A* Algorithm
def AStarSearch(graph, goalName, startName):
    solution = PriorityQueue() #priority queue for solution
    with solution.mutex:
        solution.queue.clear() #make sure we always start
```

```
on an empty queue

    graph.removeAllNodeParent() #make sure all the nodes
also has empty parents

    solution2 = [] #array for solution 2

    goalNode = graph.searchByName(goalName) # goalNode
    startNode = graph.searchByName(startName) # startNode
    solution.put((0, startNode))
    i = -1
    #print(curNode[1].name)
    while(not is_in_queue(goalNode, solution) and
i<len(solution.queue)):
        i+=1
        if (i>=len(solution.queue)): #protection againts
not found
            break
        curNode = solution.queue[i]

        for j in range (0,i): #guard for elements that are
inserted before curNode
            for key,value in
solution.queue[j][1].neighbors.items():
                if (not is_in_queue(key, solution)):
                    key.addParent((solution.queue[j])[1])

solution.put((key.calculateHaversine(goalNode)+
value+key.getDistanceBetween(goalNode), key))

        for key,value in curNode[1].neighbors.items():
            if (not is_in_queue(key, solution)):
                key.addParent(curNode[1])

solution.put((key.calculateHaversine(goalNode)+ value,
key)) #the a star portion of the code

        solution.queue.sort()

    # Heuristic, if iteration count is bigger than the size
of the graph, no solution is available
    # Not found
    if (i>=len(solution.queue)):
        return []

    finalGoalNode = findInQueue(goalNode,solution)  #get
```

```
the final goal node (with the parent)

    #print(finalGoalNode.getParents())
    while(finalGoalNode.hasParents()):
        solution2.append(finalGoalNode)
        finalGoalNode =
findInQueue(finalGoalNode.getParents(),solution)
    #endwhile
    solution2.append(finalGoalNode)
    solution2.reverse()
    #solution2 is now the array containing the path for the
graph

    return solution2

# Read file and create a graph
def getGraphFromFile():
    # Init boolean to flag if file is found
    foundFile = False

    # Repeat until a file is loaded successfully
    while(not foundFile):
        try:
            # Create a graph based on the files
            filename = input("Enter the name of the file
you want to read: ")
            graph = Graph(filename)
            foundFile = True
        except:
            print("No file found!\n")

    return graph
    #graph.checkGraph()

def visualize(graph, miscList):
    check = int(input("Choose 1 to visualize with NetworkX,
2 with HERE Maps API, and 3 for to use both sequentially:
"))
    if(check == 1):
        # Visualize with NetworkX
        graph.visualize(miscList)
    elif(check == 2):
        # Visualize with HERE Map API through Flask
        print("Follow the local link to see HERE Map API
Visualization!")
        print("Use CTRL-C to restart!\n")
```

```
        app.run(debug = False)
    elif(check == 3):
        # NetworkX and HERE
        graph.visualize(miscList)
        print("Follow the local link to see HERE Map API
Visualization!")
        print("Use CTRL-C to restart!\n")
        app.run(debug = False)
    else:
        print("Invalid Input! Canceling Visualization!")

# Init Flask for HERE MAPS API
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def map():
    # Get points
    points = graph.getPoints()

    # Get solution from A* Search
    solution = AStarSearch(graph, goalName, startName)

    # To be able to pass into JS, the elements shouldn't be
object
    # Insert informations about the node to the path list
    path = []
    for node in solution:
        sols = []
        sols.append(node.x)
        sols.append(node.y)
        sols.append(node.name)
        path.append(sols)

    # Render the template of map and pass the lists
    return render_template('map.html', points = points,
path = path)

#-------------------------------------------------------------
------------------#
# Main Program

# Init read file
graph = getGraphFromFile()

# Visualize the full graph with NetworkX
graph.visualize([])
```

```
#graph.checkGraph()

# Repeat until the user wants to exit the program
while(True):

    # Request input from the user
    print()
    choice = int(input("Enter 1 to proceed to A*, 2 to
reinput the file, 3 to exit the program: "))

    # Commence A* Processes
    if(choice == 1):
        print("Entering A* Process...\n")
        graph.showAllPlaces()
        startName = input("Please enter the start node: ")
# Get start name of the node
        goalName = input("Please enter the goal node: ")
# Get goal name of the node

        if (not graph.isNodeInGraphByName(startName) or
(not graph.isNodeInGraphByName(goalName))): # Node(s) is
not in graph
            print("Sorry, the node(s) you are looking for
is not in the graph!")
        elif(startName == goalName):
            print("You entered same names. We cannot
move!") # Same start and goal node
        else:
            solution = AStarSearch(graph, goalName,
startName) # Find The Closest Path with A Star

            if (len(solution) == 0): # Check if solution
not found
                print ("NO PATH IS FOUND")
            else:
                # Print the path
                print("FOUND A PATH!")
                print("Closest path from", startName, "to",
goalName, "is:")
                for nodes in solution[:-1]:
                    print(nodes.name, end = " => ")
                print(solution[-1].name)
                print()

                visualize(graph, solution)
```

```
    # Commence read file again
    elif(choice == 2):
        graph = getGraphFromFile()
        graph.visualize([])
    # Exit the program
    elif(choice == 3):
        print("Thank you! !_!")
        exit()
    # Invalid choices
    else:
        print("Invalid Input! Try again.")
```

map.html

```
<html>
<head>
    <title>Path Visualization</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0, maximum-scale=1.0, user-scalable=yes">
    <meta http-equiv="Content-type"
content="text/html;charset=UTF-8">
    <link rel="stylesheet" type="text/css"
href="https://js.api.here.com/v3/3.1/mapsjs-ui.css" />
    <script type="text/javascript"
src="https://js.api.here.com/v3/3.1/mapsjs-core.js"></scrip
t>
    <script type="text/javascript"
src="https://js.api.here.com/v3/3.1/mapsjs-service.js"></sc
ript>
    <script type="text/javascript"
src="https://js.api.here.com/v3/3.1/mapsjs-ui.js"></script>
    <script type="text/javascript"
src="https://js.api.here.com/v3/3.1/mapsjs-mapevents.js"></
script>
</head>

<body style='margin: 0'>
<div style="width: 100%; height: 100%"
id="mapContainer"></div>
<script type="text/javascript">
    // Init Points and Path in the map from Flask
    // Parse from Flask to JSON
    // let points = [[-6.891594303998672,
107.61069133183867], [-6.893408313362439,
107.61061071943095],
```

```
    //           [-6.892874781942925, 107.61285890989026],
[-6.8899937016911, 107.61295743619061],
    //           [-6.887463546266129, 107.60874522288952],
[-6.8902251567611845, 107.60685337297433],
    //           [-6.893800613850137, 107.61374151753564],
[-6.896287515528555, 107.609195095660381]
    var points = JSON.parse('{{points|tojson}}');
    var path = JSON.parse('{{path|tojson}}');
</script>
<script type="text/javascript">
    // Add marker to group
    // On coordinate and with locationInfo as the
information
    function addMarkerToGroup(group, coordinate,
locationInfo) {
        var marker = new H.map.Marker(coordinate);
        // Add custom pop-up information to the marker
        marker.setData(locationInfo);
        group.addObject(marker);
    }

    // Add clickable bubble on each marker
    function addClickableBubble(map, points){
        // Create a new group
        var group = new H.map.Group();

        // Add the group to the map
        map.addObject(group);

        // Use 'tap' event listener, that opens info bubble
when clicked
        group.addEventListener('tap', function (evt) {
            // Event target is the marker itself, group is
a parent event target
            // For all objects that it contains
            var bubble =  new
H.ui.InfoBubble(evt.target.getGeometry(), {
                // Read custom data
                content: evt.target.getData()
            });
            // Show info bubble
            ui.addBubble(bubble);
        }, false);

        // Add the marker for each nodes
        points.forEach(node => {addMarkerToGroup(group,
```

```
{lat: node[0], lng: node[1]}, node[2])});
    }

    // Add Polyline to draw the path
    function addPolylineToMap(map, path) {
        // Init Linestring
        var lineString = new H.geo.LineString();

        // Add polyline for each points in the path
        path.forEach(node => {lineString.pushPoint({lat:
node[0], lng: node[1]})});

        map.addObject(new H.map.Polyline(
            lineString, {style: {lineWidth: 3}}
        ));
    }

    // Add markers to each points in the map
    function addMarkerToPoints(map, points){
        points.forEach(node => {
            var marker = new H.map.Marker({lat: node[0],
lng: node[1]});

            // Add the marker to the map:
            map.addObject(marker);
        });
    }

    // Add Info Bubble as Information of the markers
    function addInfoBubble(map, points){
        points.forEach(node => {
            var bubble = new H.ui.InfoBubble({lat: node[0],
lng: node[1]}, {
                        content: node[2]
            });
            // Add info bubble to the UI:
            ui.addBubble(bubble)});
    }

    // Initialize communication with the platform through
HERE Maps API Key
    var platform = new H.service.Platform({
        apikey:
'O6p0lXfBGg9XH8480E9i_ga3s3peFP73YnFpLjl-WKc'
    });
```

```
    // Create Default Layers for the platform
    var defaultLayers = platform.createDefaultLayers();

    // Initialize the map with the centered value is the
first node in points
    var map = new
H.Map(document.getElementById('mapContainer'),
defaultLayers.vector.normal.map,
        {center: {lat: points[0][0], lng: points[0][1]},
        zoom: 16,
        pixelRatio: window.devicePixelRatio || 1
    });

    // Add a resize listener to make sure that the map
occupies the whole container
    window.addEventListener('resize', () =>
map.getViewPort().resize());

    // Make the map interactive
    // MapEvents enables the event system
    // Behavior implements default interactions for
pan/zoom (also on mobile touch environments)
    var behavior = new H.mapevents.Behavior(new
H.mapevents.MapEvents(map));

    // Create the default UI components
    var ui = H.ui.UI.createDefault(map, defaultLayers);

    // Use the functions defined above to add properties to
the map for visualizations
    addClickableBubble(map, points);
    addPolylineToMap(map, path);
    // addMarkerToPoints(map, points);

</script>
</body>
</html>
```

2. Peta atau Graf Input

Berikut merupakan beberapa berkas *testcase* yang digunakan di dalam pengujian program.

testcase.txt

```
8
InstitutTeknologiBandung -6.891594303998672
107.61069133183867
TamanGanesha -6.8934908803877395 107.61059221679027
GeprekBensu -6.892713359686696 107.61287165119809
SPBU -6.889986620479006 107.61290383770567
Tamansari -6.887334425445851 107.60875177820569
Bonbin -6.89019964748707 107.60690641843756
Diskominfo -6.8962623261680775 107.60920158762804
NoahsBarn -6.887049987581403 107.61280227515437
0 1 0 0 1 1 0 1
1 0 1 0 0 1 1 0
0 1 0 1 0 0 0 0
0 0 1 0 0 0 0 1
1 0 0 0 0 1 0 1
1 1 0 0 1 0 1 0
0 1 0 0 0 1 0 0
1 0 0 1 1 0 0 0
```

testcase2.txt

```
10
AlunAlunBandung -6.92182559249719 107.60695420607672
MuseumAsiaAfrika -6.921189604928431 107.60952504581432
PLNBandung -6.920880652884007 107.60837577208487
Cikapundung -6.919081870809035 107.60892135022254
KantorPos -6.920667768326371 107.60619855988753
MasjidRayaBandung -6.921681840605713 107.60607795650104
PendopoBandung -6.9233941110536215 107.60698555328437
BankPanin -6.919826236345531 107.60679326883204
GrandYogyaKepatihan -6.923618925963251 107.60572256674298
Cafe67 -6.922399953684318 107.60865480249316
0 0 1 0 0 1 1 1 0 1
0 0 1 0 0 0 0 0 0 1
1 1 0 1 1 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0
0 0 1 0 0 1 0 1 0 0
1 0 0 0 1 0 0 1 1 0
1 0 0 0 0 0 0 0 1 1
```

```
1 0 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0
1 1 0 0 0 0 1 0 0 0
```

testcase3.txt

```
8
myHouse -6.3441000539837615 107.15762243971379
posSatpam -6.343113046621595 107.15554640995084
pomBensin -6.343314313629815 107.15520040499138
theHarvest -6.343801013744161 107.15494207962544
abubaSteak -6.343398480022214 107.15423665861792
jlnKeludRaya -6.342334156903663 107.15566451270803
redDoorz -6.341657014490506 107.15588706158087
indomaret -6.344354191621068 107.1557430069852
0 1 0 0 0 1 1 0
1 0 1 1 1 1 0 1
0 1 0 1 1 0 0 1
0 1 1 0 1 0 0 1
0 1 1 1 0 0 0 1
1 1 0 0 0 0 1 0
1 1 0 0 0 1 0 0
0 1 1 1 1 0 0 0
```

testcase4.txt

```
9
MetroIndahMall -6.942120717034387 107.65874719557186
JlSoekarnoHatta -6.940290411439278 107.65824797384171
JembatanSungaiCicadas -6.942040661767213 107.65277526794446
YamahaServiceCenter -6.943154221380146 107.65990485388869
JlJupiterBarat26 -6.943648609245794 107.65761911899006
TamanS2 -6.945294809444772 107.65724156585514
TamanJupiter -6.9440854448675 107.66042481163986
JlJupiterBaratUtama -6.944109165356709 107.66130417588958
JlRayaCirebonBandung -6.939199773655335 107.66391628016386
0 1 0 1 1 0 0 0 1
1 0 1 1 0 0 0 0 0
0 1 0 0 0 0 0 0 0
1 1 0 0 1 0 1 0 0
1 0 0 1 0 1 0 0 0
0 0 0 0 1 0 0 0 0
```

```
0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 1 0 1
1 0 0 0 0 0 0 1 0
```

testcase5.txt

```
8
MasjidAkbar -6.155596454799754 106.85367280900618
HaltePumaRaya -6.1557251235936015 106.85096449092164
MonumenOndelOndel -6.159923146837551 106.85188711889913
GrandPalace -6.159665823160958 106.85337530710886
BaksoPakPur -6.158048377736383 106.85268204886337
ApartemenPuri -6.157055851616249 106.85251566823723
WarungWonokromo -6.15677095133923 106.85388369426994
MediteraniaResidence -6.15469399578285 106.8529316210351
0 1 0 0 0 0 0 0
1 0 1 0 0 1 0 0
0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 1 0 0
0 1 0 0 1 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
```

testcase6.txt

```
8
CarrefourMedanFair 3.5913 98.66331
SekipSimpangGatsu 3.59306 98.66769
PerempatanGatsu 3.59341 98.66882
AdamMalikSimpangMayang 3.5947 98.66889
PizzaHutAdamMalik 3.59562, 98.66895
KalamKudusMedan 3.59476 98.6668
SekipSimpangMayang 3.59476, 98.66643
UnpriSekip 3.59655 98.6652
0 1 0 0 0 0 0 0
1 0 1 0 0 0 1 0
0 1 0 1 0 0 0 0
0 0 1 0 1 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 1 0
0 1 0 0 0 1 0 1
0 0 0 0 0 0 1 0
```

Baris pertama merupakan jumlah dari simpul yang ada di dalam graf. Sebanyak simpul tersebut, akan didefinisikan titik tertentu dengan nama yang tidak dipisah, serta *DD Coordinates* berupa *longitude* dan *latitude* dari titik tersebut. Terakhir, berkas ini juga disertai dengan sebuah matriks berukuran NxN, dengan N adalah jumlah simpul, yang menyatakan hubungan ketetanggaan antara setiap simpulnya.

3. *Screenshot* Peta atau Graf Hasil Eksekusi Program

```
C:\Users\User\Downloads\current>python tucil3.py
```

Gambar 1 Eksekusi Program pada Direktori Program

```
C:\Users\User\Downloads\current>python tucil3.py
Enter the name of the file you want to read: testcase.txt
```

```
Enter the name of the file you want to read: C:\Users\User\Downloads\current\test\testcase.txt
```

Gambar 2 Memasukkan Nama *Source File* untuk Graf

```
Enter the name of the file you want to read: C:\Users\User\Downloads\current\test\testcase.txt

Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program: 1
Entering A* Process...

List of available places:
ITB
TamanGanesha
GeprekBensu
SPBU
Tamansari
Bonbin
Diskominfo
NoahsBarn

Please enter the start node:
```

Gambar 3.1 Input Pilihan Eksekusi Proses: Proses Pencarian A*

```
Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program: 2
Enter the name of the file you want to read:
```

Gambar 3.2 Input Pilihan Eksekusi Proses: Input File Baru

```
Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program: 3
Thank you! !_!
```

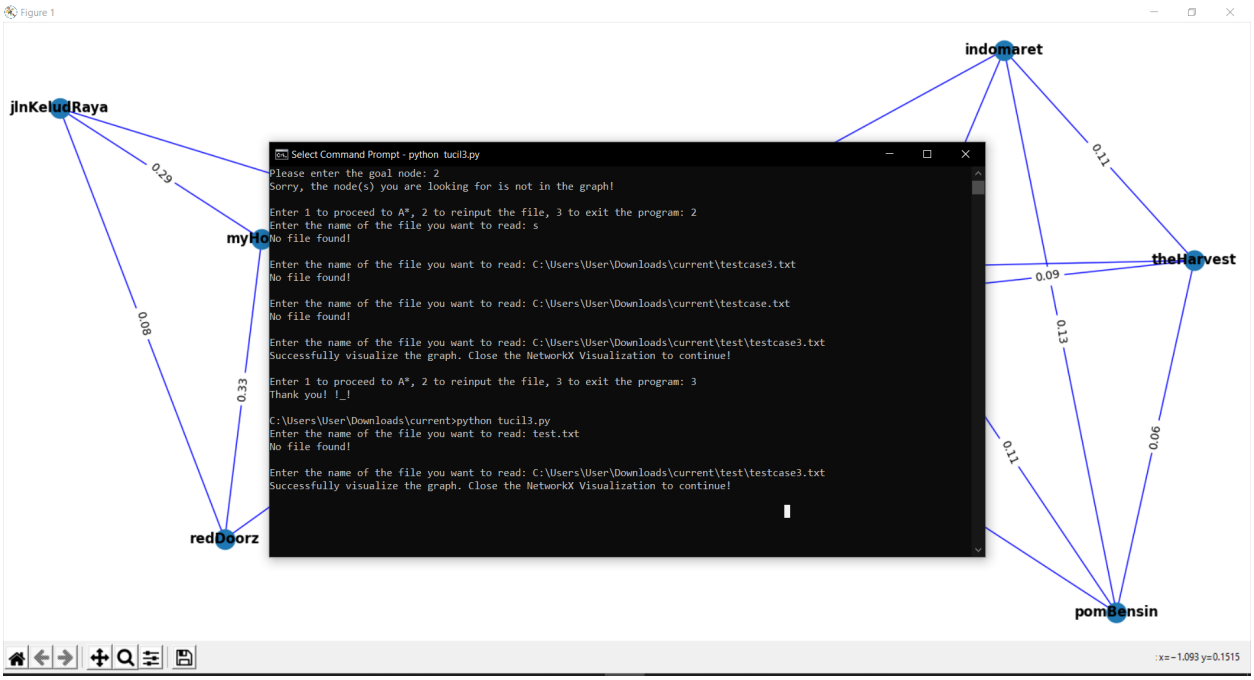
Gambar 3.3 Input Pilihan Eksekusi Proses: Keluar dari Program

```
Enter the name of the file you want to read: test.txt
No file found!

Enter the name of the file you want to read: _
```

Gambar 4 File Tidak Valid atau Ditemukan

```
Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program: 4
Invalid Input! Try again.

Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program:
```

Gambar 5 Pilihan yang Tidak Valid



Gambar 6 Visualisasi Awal Jika *File* Berhasil Dibaca

```
Enter 1 to proceed to A*, 2 to reinput the file, 3 to exit the program: 1
Entering A* Process...

List of available places:
ITB
TamanGanesha
GeprekBensu
SPBU
Tamansari
Bonbin
Diskominfo
NoahsBarn

Please enter the start node: ITB
Please enter the goal node: SPBU
```

Gambar 7 Tampilan Input untuk Proses Pencarian A*

```
Please enter the start node: ITB
Please enter the goal node: SPBU
FOUND A PATH!
Closest path from ITB to SPBU is:
ITB => TamanGanesha => GeprekBensu => SPBU
```

Gambar 8 Tampilan Jalur Hasil Pencarian A* dalam Bentuk Teks

```
Please enter the start node: ITB
Please enter the goal node: ITB
You entered same names. We cannot move!
```

Gambar 9 Tampilan Program jika Input Lokasi yang Dimasukkan Sama

```
Please enter the start node: itb
Please enter the goal node: ITB
Sorry, the node(s) you are looking for is not in the graph!
```

Gambar 10 Tampilan Program jika Input Lokasi Tidak Ada di Graf (*Case Sensitive*)

```
Please enter the start node: WarungWonokromo
Please enter the goal node: MediteraniaResidence
NO PATH IS FOUND
```

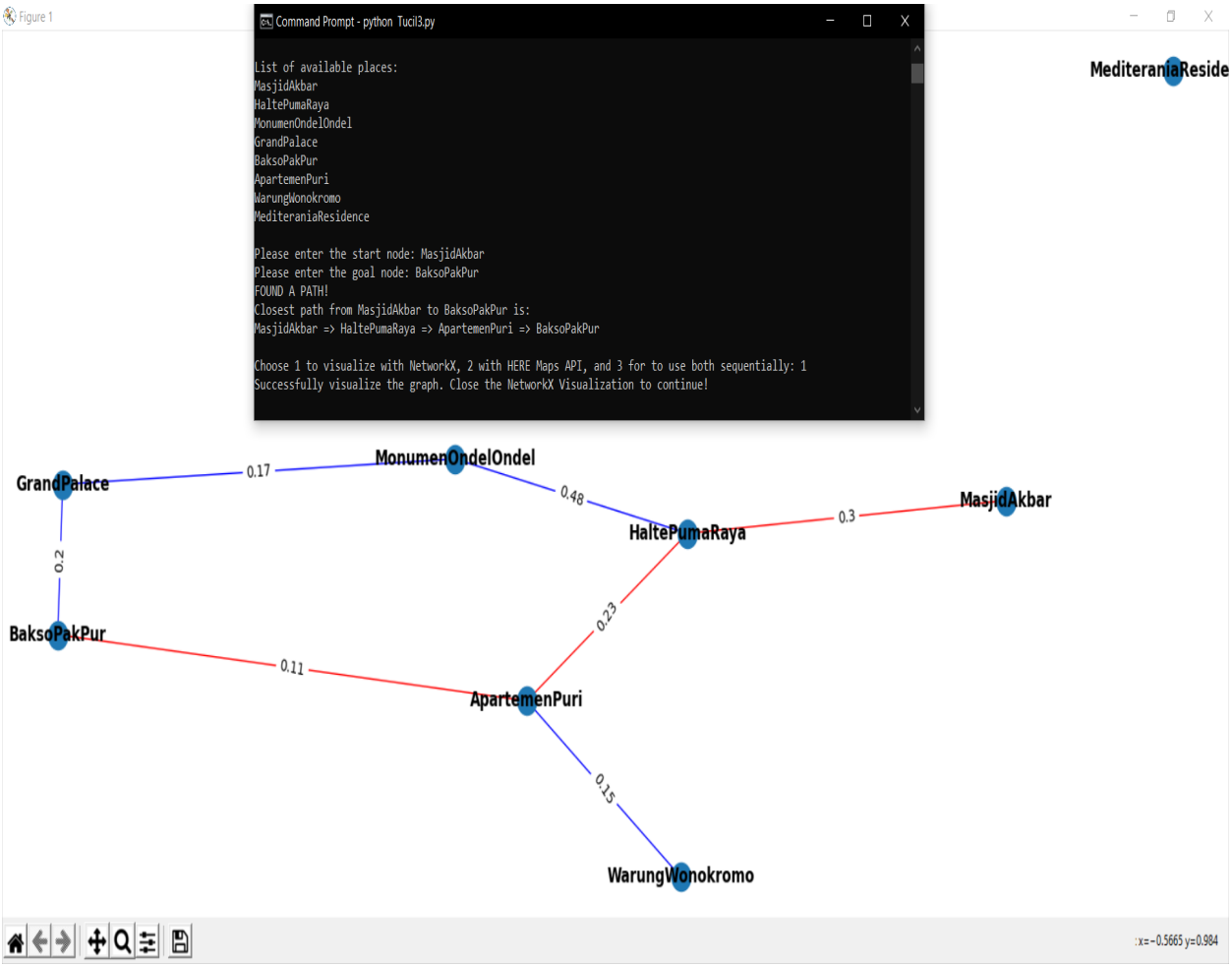
Gambar 11 Tampilan Program jika A* Tidak Menghasilkan Solusi untuk *Testcase* 5

```
Please enter the start node: MasjidAkbar
Please enter the goal node: BaksoPakPur
FOUND A PATH!
Closest path from MasjidAkbar to BaksoPakPur is:
MasjidAkbar => HaltePumaRaya => ApartemenPuri => BaksoPakPur

Choose 1 to visualize with NetworkX, 2 with HERE Maps API, and 3 for to use both sequentially: 1
```

Gambar 12 Tampilan Program saat Meminta Input Pilihan Visualisasi

```
Choose 1 to visualize with NetworkX, 2 with HERE Maps API, and 3 for to use both sequentially: 4
Invalid Input! Canceling Visualization!
```

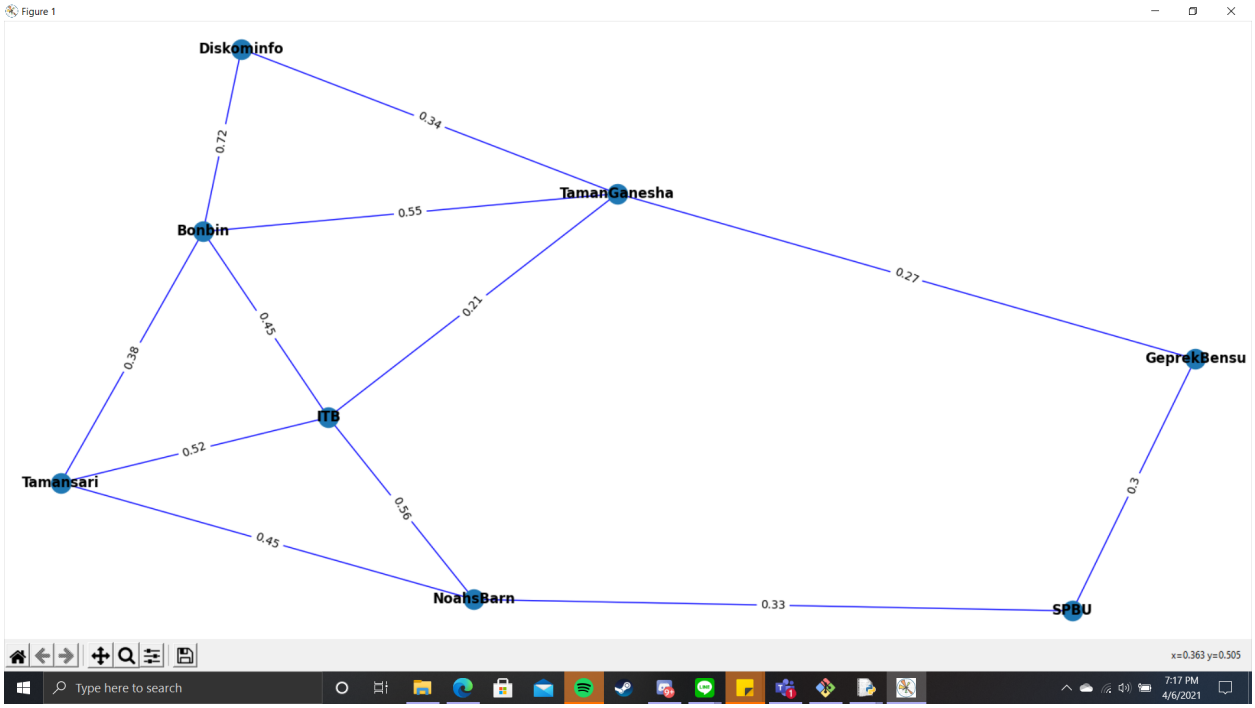
Gambar 13 Tampilan Program saat Pilihan Visualisasi Tidak Valid



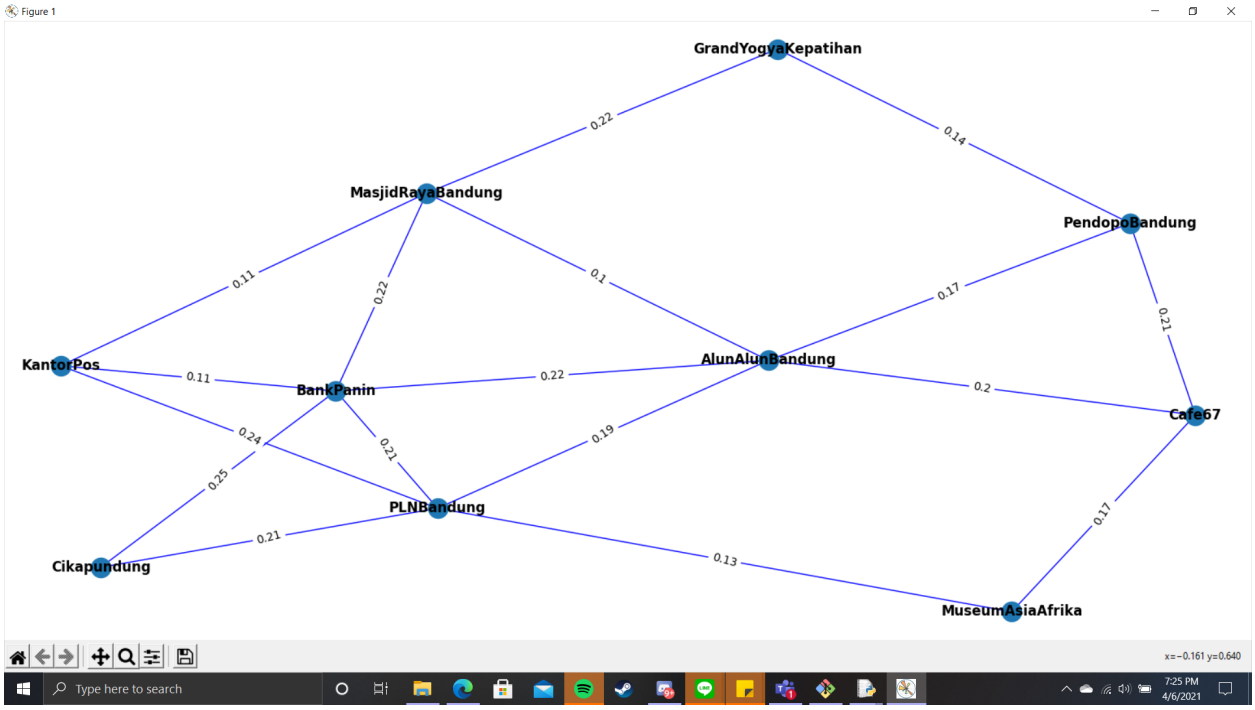
Gambar 14 Visualisasi Program dengan NetworkX

```
Choose 1 to visualize with NetworkX, 2 with HERE Maps API, and 3 for to use both sequentially: 2
Follow the local link to see HERE Map API Visualization!
Use CTRL-C to restart!

 * Serving Flask app "Tucil3" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

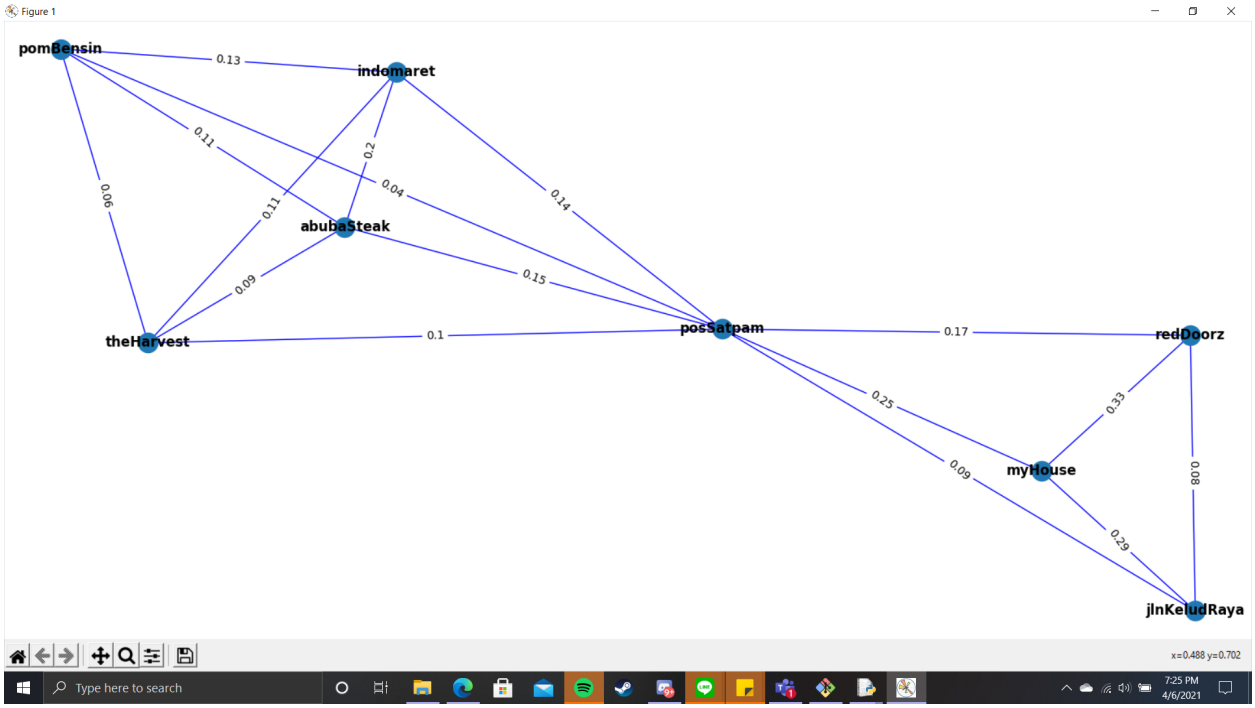
Gambar 15 Tampilan Program saat Memilih Visualisasi dengan HERE Maps API



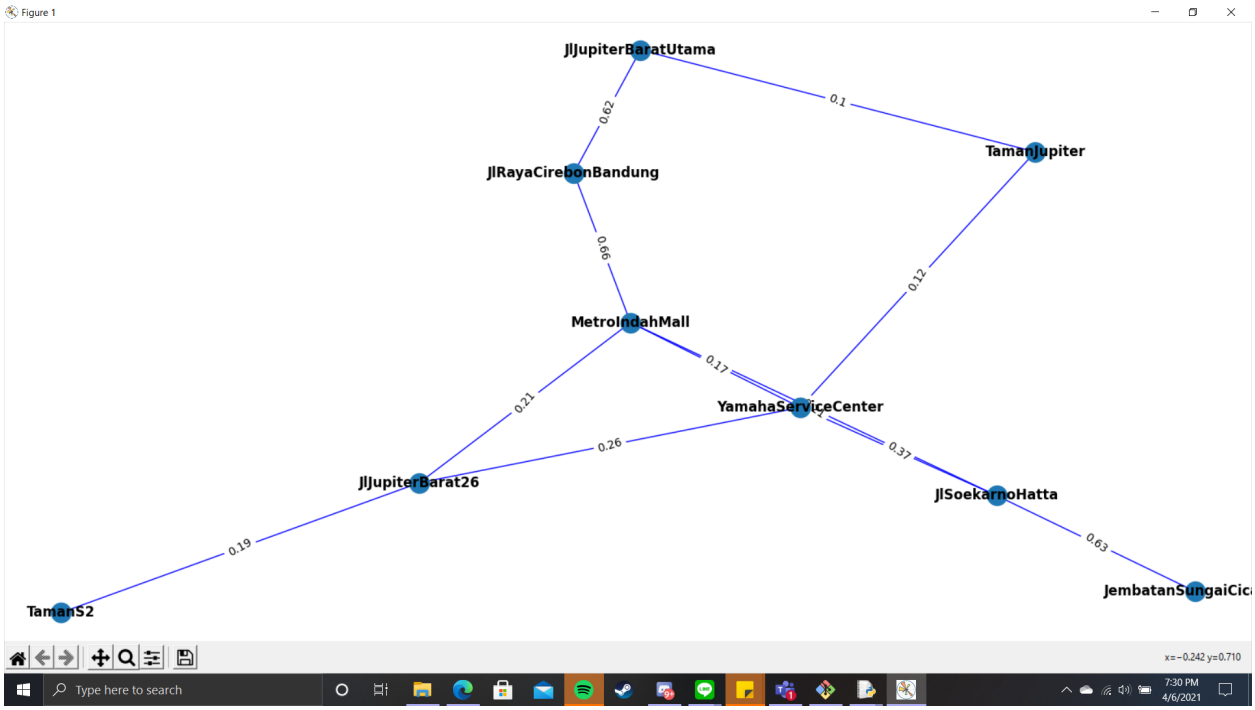
Gambar 16 Visualisasi Graf untuk *Testcase* 1: Area Institut Teknologi Bandung



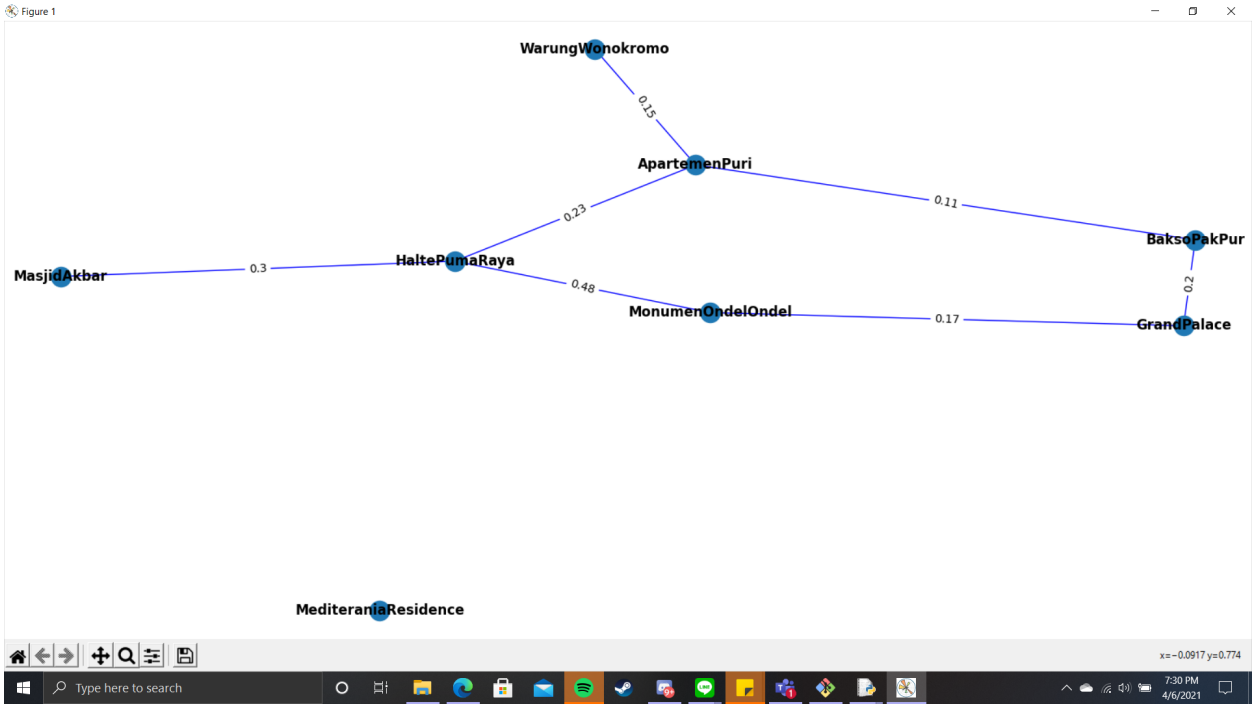
Gambar 17 Visualisasi Graf untuk *Testcase* 2: Area Alun-Alun Bandung



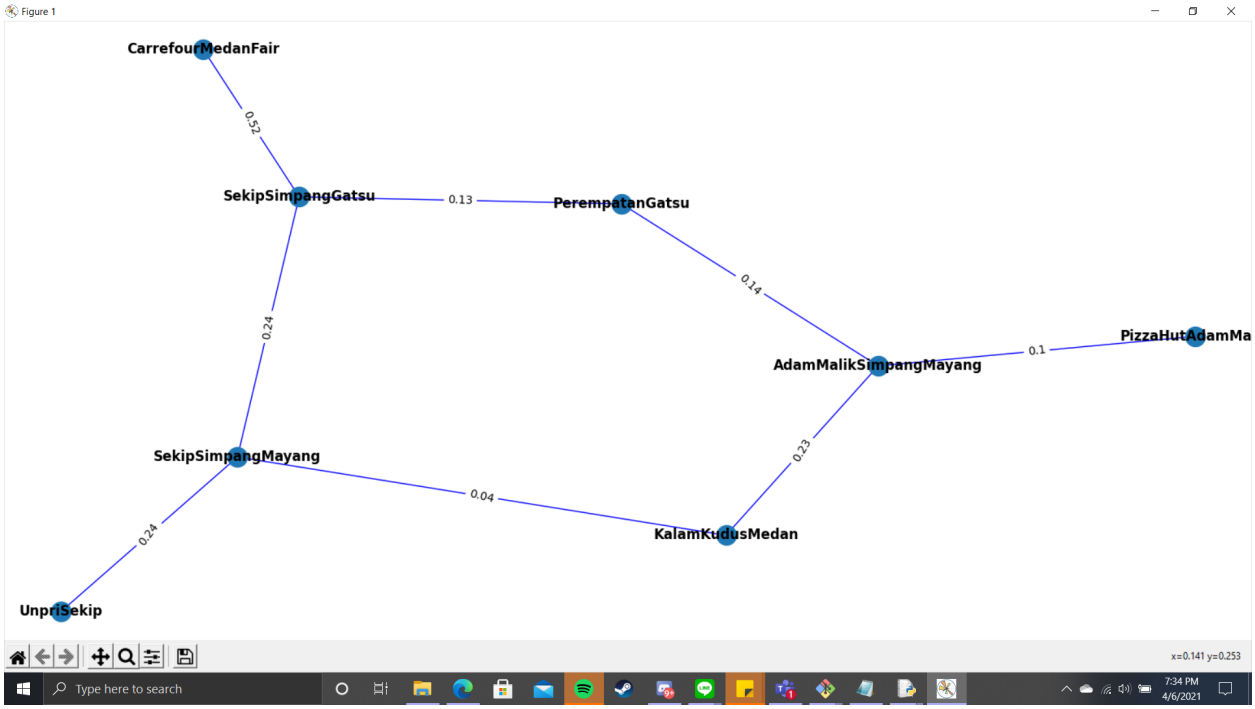
Gambar 18 Visualisasi Graf untuk *Testcase* 3: Area Rumah Hizkia



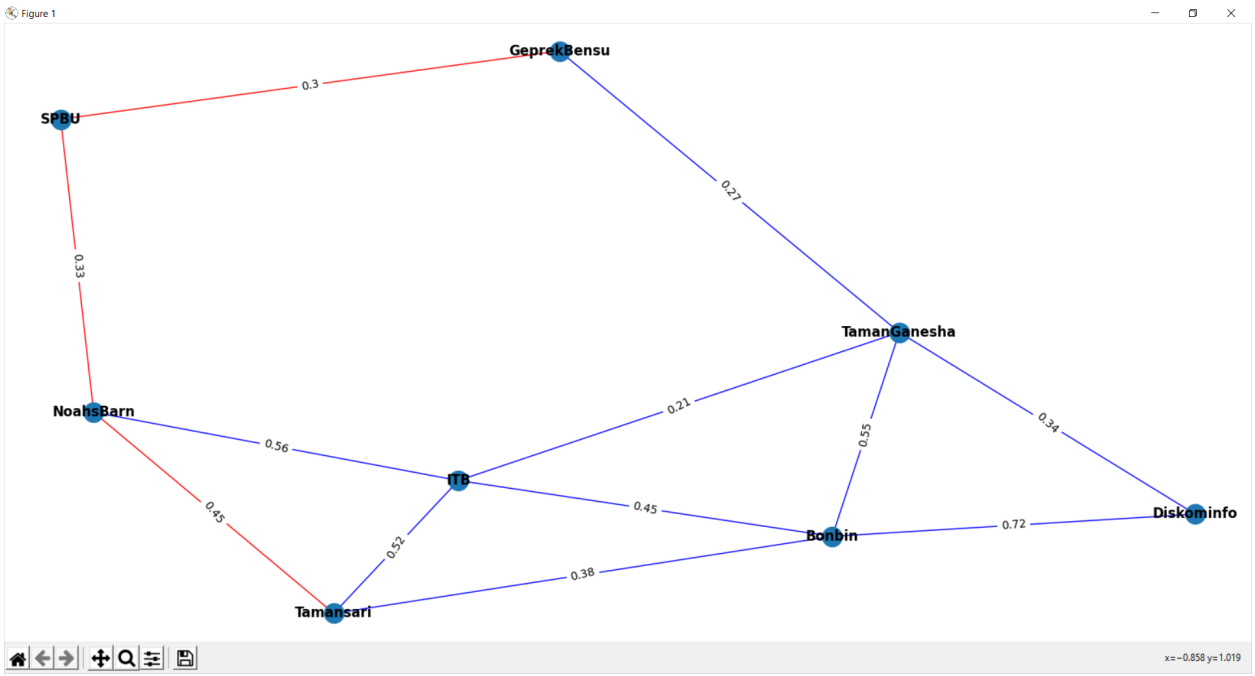
Gambar 19 Visualisasi Graf untuk *Testcase* 4: Area Buah Batu



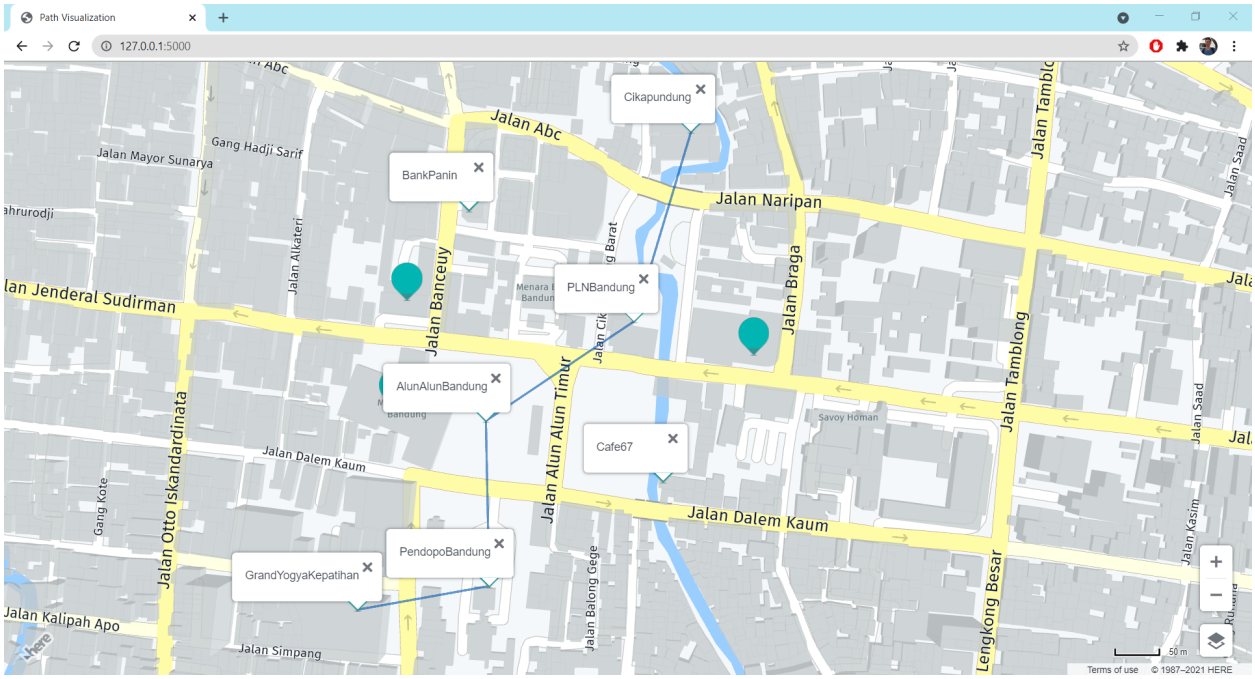
Gambar 20 Visualisasi Graf untuk *Testcase* 5: Area Masjid Akbar



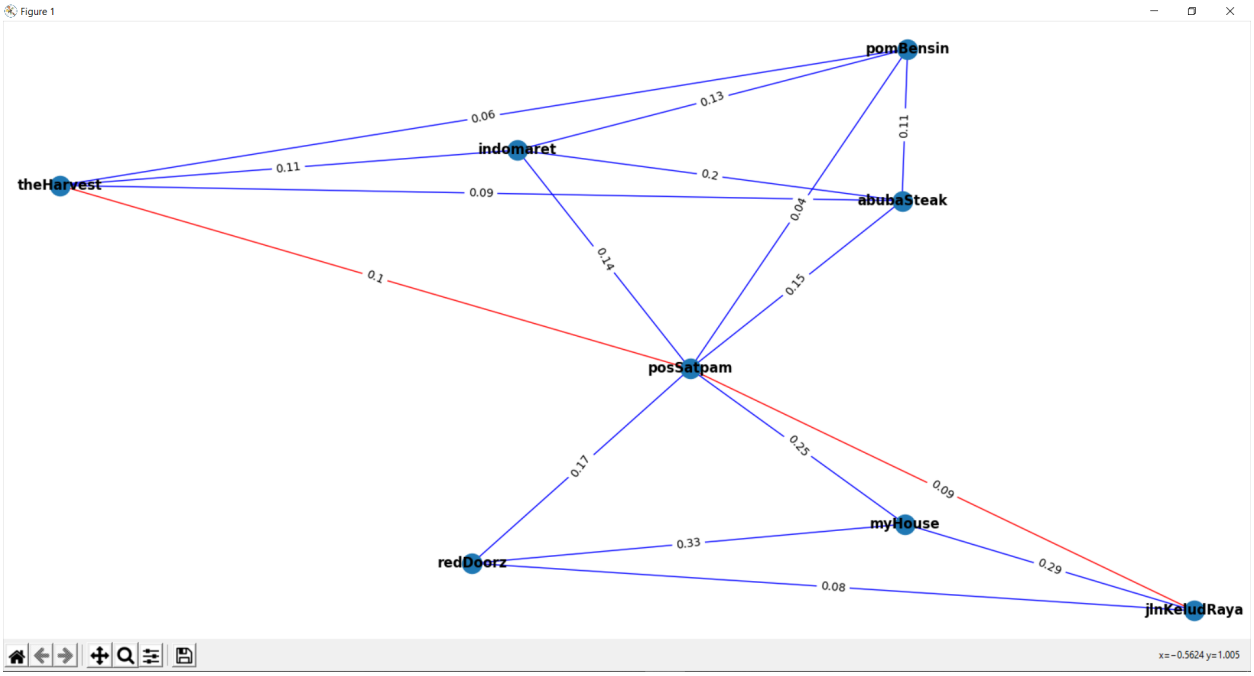
Gambar 21 Visualisasi Graf untuk *Testcase* 6: Area *Carrefour Medan Fair*



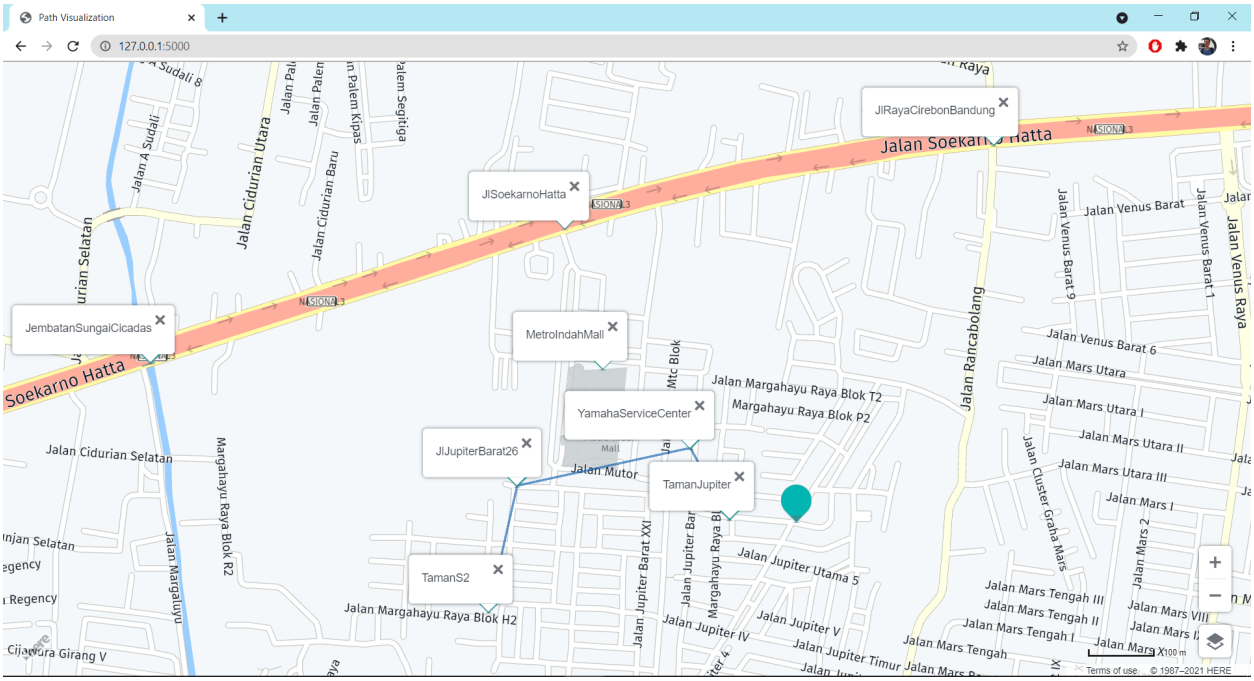
Gambar 22 Visualisasi A* dari GeprekBensu ke Tamansari untuk *Testcase* 1



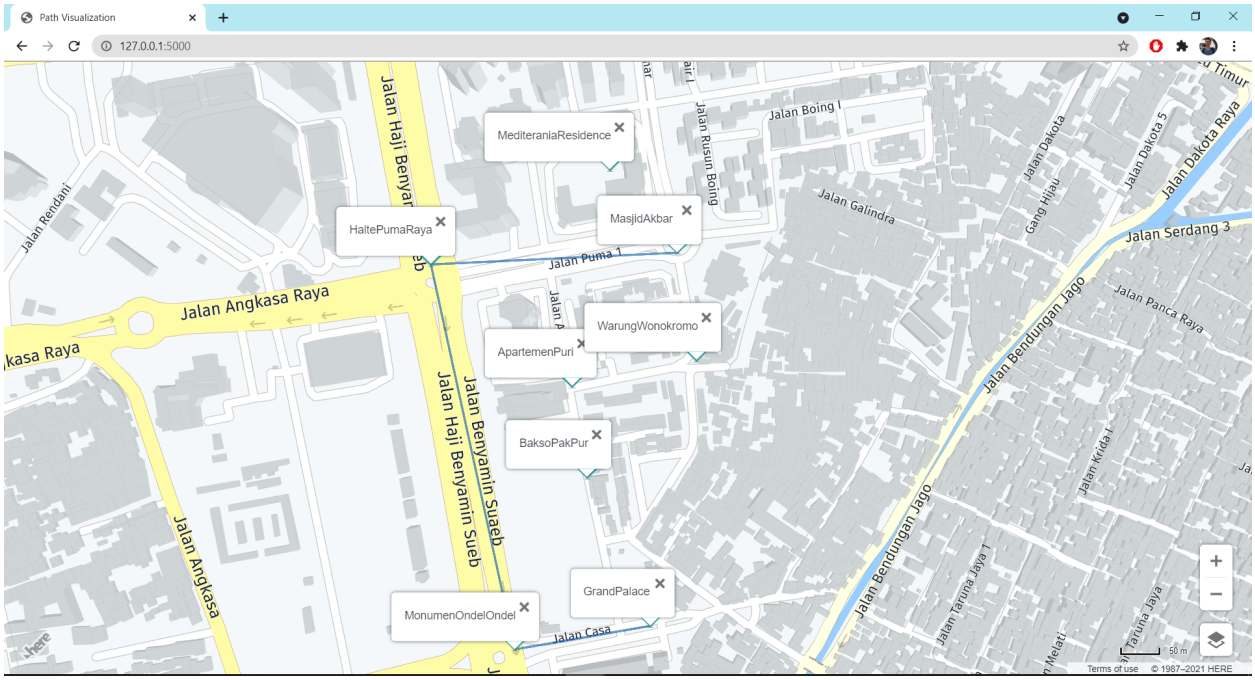
Gambar 23 Visualisasi A* dari Cikapundung ke GrandYogyaKepatihan untuk *Testcase* 2



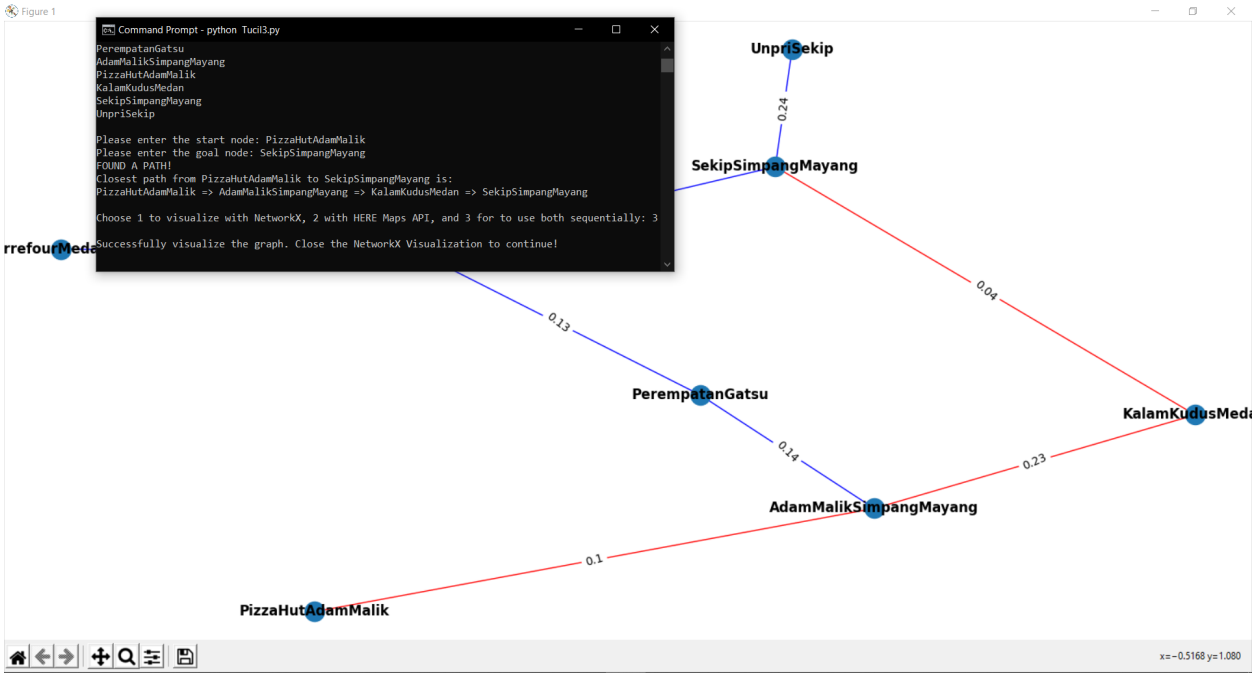
Gambar 24 Visualisasi A* dari jlnKeludRaya ke theHarvest untuk *Testcase* 3



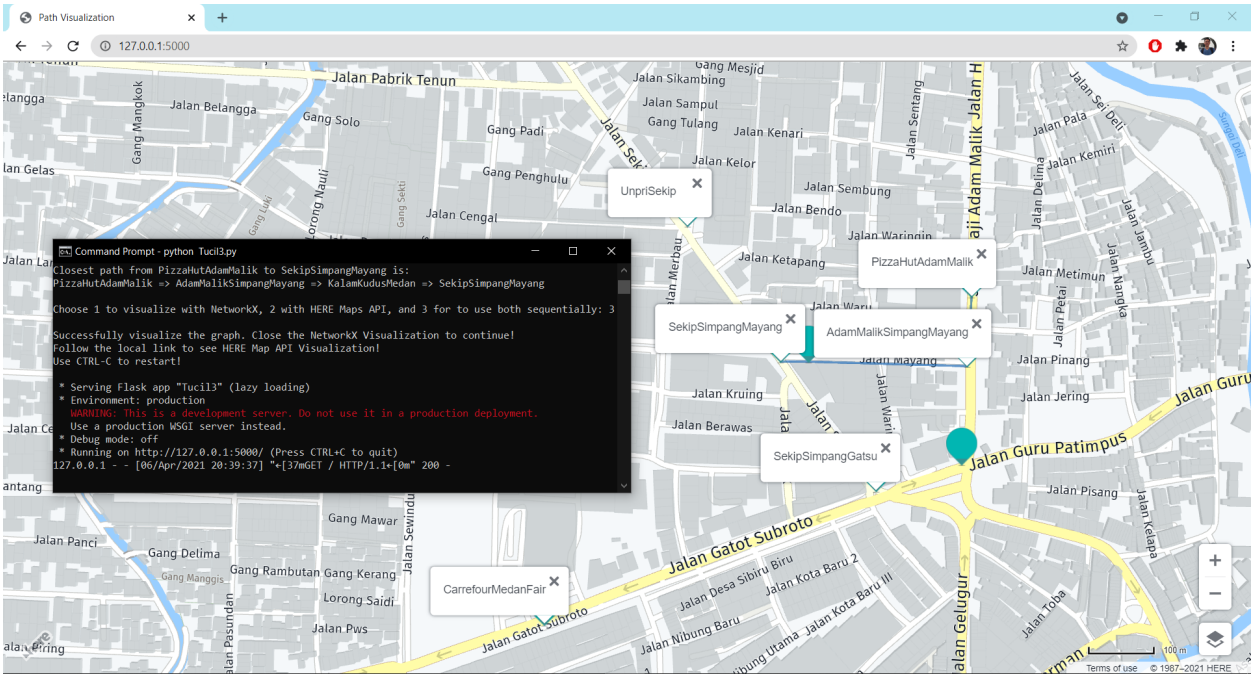
Gambar 25 Visualisasi A* dari TamanJupiter ke TamanS2 untuk *Testcase* 4



Gambar 26 Visualisasi A* dari MasjidAkbar ke GrandPalace untuk *Testcase* 5



Gambar 27.1 Eksekusi Pilihan 3 dengan Visualisasi A* Menggunakan NetworkX dan HERE Maps API Sekaligus dari PizzaHutAdamMalik ke SekipSimpangMayang untuk *Testcase* 6



Gambar 27.2 Eksekusi Pilihan 3 dengan Visualisasi A* Menggunakan NetworkX dan HERE Maps API Sekaligus dari PizzaHutAdamMalik ke SekipSimpangMayang untuk *Testcase* 6

4. *Link* Kode Program

Berikut merupakan *link* repositori Github yang mengandung *source code* program ini.

https://github.com/HizRadit07/tucil3-stima

5. Lampiran

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	-
5	Bonus: Program dapat menerima input tempat yang ingin divisualisasikan dari *file*, dan melakukan visualisasi graf dan peta serta jalur terpendek menggunakan NetworkX dan HERE Maps API sebagai alternatif dari Google Maps API. Peta yang ditampilkan bisa diklik untuk menampilkan informasi lokasi terkait tempat tersebut.	✓