TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA 2022/2023

Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc

Disusun oleh:

Rayhan Hanif Maulana Pradana/13521112

Irgiansyah Mondo/13521167

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

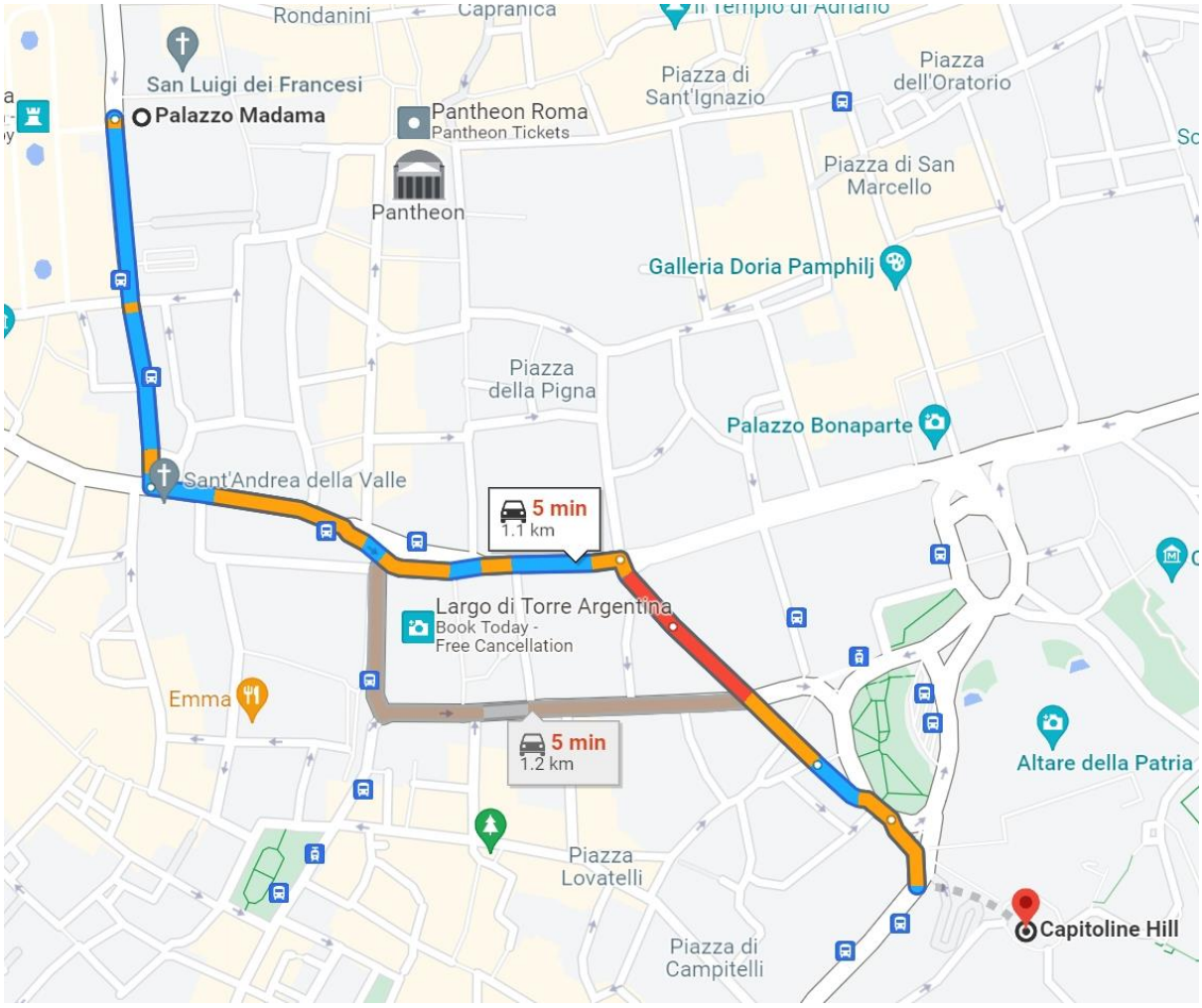
INSTITUT TEKNOLOGI BANDUNG

2022/2023

BAB I

DESKRIPSI MASALAH

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Salah satu penerapan dari implementasi Algoritma UCS dan Algoritma A* adalah penentuan rute terdekat pada sebuah peta. Peta atau graf dengan titik-titik (node) dapat direpresentasikan dengan matriks ketetanggaan berbobot, dimana elemen dari matriks ketetanggaan tersebut merupakan jarak sebenarnya dari suatu node ke node yang lainnya.



Contoh penentuan rute terdekat pada Google Maps

BAB II

ALGORITMA *PATH FINDING*

2.1 Algoritma Uniform Cost Search (UCS)

Berikut adalah langkah-langkah dari penentuan rute terdekat dengan menggunakan algoritma UCS :

1. Buat priority queue berdasarkan urutan cost/jarak terkecil yang ditempuh sejauh ini
2. Kunjungi simpul awal/start node
3. Cek semua tetangga dari simpul awal dan enqueue simpul-simpul tetangga tersebut ke priority queue
4. Dequeue priority queue dan kunjungi node hasil dequeue tersebut
5. Cek semua tetangga dari simpul yang sedang dikunjungi dan enqueue simpul-simpul tetangga tersebut ke priority queue
6. Ulangi langkah 5 dan 6 hingga ditemukan simpul tujuan/end node

2.2 Algoritma A* (A Star)

Implementasi algoritma A* yang dibuat dapat menggunakan salah satu dari dua heuristik :

1. Heuristik berdasarkan Euclidean Distance
   Euclidean distance merupakan jarak garis lurus antara satu titik ke titik lain pada bidang koordinat kartesius

   $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

   Sehingga Euclidean Distance baik untuk titik dengan koordinat kartesius ataupun titik dengan koordinat geographical dengan jarak garis lurus relatif kecil

2. Heuristik berdasarkan Haversine Distance
   Haversine distance merupakan jarak garis lurus antara satu titik ke titik lain dengan memperhitungkan lingkar pada sebuah benda bulat. Haversine distance menggunakan koordinat geographical dengan latitude (ϕ) dan longitude (λ), serta radius bola (r), yang dalam kasus ini adalah radius bumi.

   $$\begin{aligned} d &= 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)}\right) \\ &= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \left(1 - \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) - \sin^2\left(\frac{\varphi_2 + \varphi_1}{2}\right)\right) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \\ &= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right). \end{aligned}$$

   Sehingga Haversine Distance baik untuk titik dengan koordinat geographical dengan jarak garis lurus relatif besar

Berikut adalah langkah-langkah dari penentuan rute terdekat dengan menggunakan algoritma A* :

1. Buat priority queue berdasarkan urutan heuristik dari terkecil. Nilai heuristik adalah cost/jarak yang ditempuh sejauh ini ditambah dengan metode penentuan garis lurus yang dipakai (Euclidean Distance atau Haversine Distance)
2. Kunjungi simpul awal/start node
3. Cek semua tetangga dari simpul awal dan enqueue simpul-simpul tetangga tersebut ke priority queue
4. Dequeue priority queue dan kunjungi node hasil dequeue tersebut
5. Cek semua tetangga dari simpul yang sedang dikunjungi dan enqueue simpul-simpul tetangga tersebut ke priority queue
6. Ulangi langkah 5 dan 6 hingga ditemukan simpul tujuan/end node

BAB III

SOURCE CODE

Implementasi dilakukan dengan menggunakan bahasa pemrograman Python3.

3.1 node.py

```
You, 9 hours ago | 1 author (You)
class Node:
    def __init__(self, name, x, y, idx):
        self.name = name
        self.x = x
        self.y = y
        self.idx = idx  # Index of the node in the adjacency matrix
        self.distance = 0   # The Euclidean Distance of this Node to another Node

    def euclideanDistance(self, otherNode):
        self.distance = math.sqrt(math.pow((self.x - otherNode.x), 2) + math.pow((self.y - otherNode.y), 2))

    def haversineDistance(self, otherNode):
        r = 6371       You, 9 hours ago • Penyesuaian A-star dengan program utama …
        body = np.sin((self.x - otherNode.x)/2)**2 + np.cos(self.x)*np.cos(otherNode.x)*(np.sin((otherNode.y - self.y)/2))**2
        self.distance = 2*r*np.arcsin(np.sqrt(body))
```

3.2 livenode.py

```
class LiveNode:
    def __init__(self, node): # node is an instance of the class Node
        self.node = node
        self.prevNodes = ""
        self.costSoFar = 0
        self.compValue = 0  # Value used for comparison in a priority queue

    def __lt__(self, otherLiveNode):
        return self.compValue < otherLiveNode.compValue

    def addPrevNode(self, prevNode): # prevNode an instance of the class LiveNode
        self.prevNodes += prevNode.node.name + ' ' + prevNode.prevNodes       You,
        self.costSoFar += prevNode.costSoFar

    def addCost(self, cost):
        self.costSoFar += cost

    def defineCompValue(self, compValue):
        self.compValue = compValue
```

3.3 main.py

```
def getNode(nodes, targetNodeName):
    for i in range(len(nodes)):
        if(nodes[i].name == targetNodeName):
            return nodes[i]
    return None


def inputNode(nodes):
    nodeFound = False
    startNode = nodes[0]
    endNode = nodes[-1]
    while(not(nodeFound)):
        startNodeInput = input("Masukkan start node : ")
        if(getNode(nodes, startNodeInput) == None):
            print("Node tidak ditemukan. Coba lagi\n")
        else:
            startNode = getNode(nodes, startNodeInput)
            nodeFound = True

    nodeFound = False
    while(not(nodeFound)):
        endNodeInput = input("Masukkan end node : ")
        if(getNode(nodes, endNodeInput) == None):
            print("Node tidak ditemukan. Coba lagi\n")
        else:
            endNode = getNode(nodes, endNodeInput)
            nodeFound = True

    return (startNode, endNode)
```

```python
def inputAlgorithm(nodes, adjacencyMatrix, startNode, goalNode):
    validAlgo = False
    while(not(validAlgo)):
        print("Algoritma : ")
        print("1. Uniform Cost Search")
        print("2. A*")
        algo = input("Pilih nomor algoritma : ")

        if(algo == '1'):
            result = ucs.uniformCostSearch(nodes, adjacencyMatrix, startNode, goalNode)
            validAlgo = True
        elif(algo == '2'):
            heuristicType = inputHeuristic()
            result = star.astar_search(nodes, adjacencyMatrix, startNode, goalNode, heuristicType)
            validAlgo = True
        else:
            print("Input invalid. Coba lagi\n")

    return result


def inputHeuristic():
    validHeuristic = False
    while(not(validHeuristic)):
        print("Tipe Heuristik : ")
        print("1. Euclidean Distance")
        print("2. Haversine Distance")
        heuristicType = input("Pilih nomor heuristic : ")

        if(heuristicType == '1' or heuristicType == '2'):
            return heuristicType
        else:
            print("Input invalid. Coba lagi\n")
```

```python
def display(nodes, adjacencyMatrix, path):
    validDisplay = False
    while(not(validDisplay)):
        print("Pilih display dengan : ")
        print("1. Network Graph")
        print("2. Map")
        displayType = input("Pilig nomor display : ")

        if(displayType == '1'):
            displaygraph.displayGraph(nodes, adjacencyMatrix, path)
            validDisplay = True
        elif(displayType == '2'):
            print('\nMembuat map . . .')
            displaymap.createMap(nodes, adjacencyMatrix, path)
            print("Map telah dibuat dengan file map.html pada root")
            validDisplay = True
        else:
            print("Input invalid. Coba lagi\n")
```

```
def main():
    fileFound = False

    while(not(fileFound)):
        filename = input("Masukkan nama file txt (bukan path & tanpa .txt) : ")
        filename = "test/" + filename + ".txt"

        if os.path.exists(filename):
            fileFound = True
            nodes = filereader.generateNodes(filename)
            adjacencyMatrix = filereader.generateAdjacencyMatrix(filename)
            startEndNode = inputNode(nodes)
            startNode = startEndNode[0]
            goalNode = startEndNode[1]

            result = inputAlgorithm(nodes, adjacencyMatrix, startNode, goalNode)

            path = ucs.constructPath(result)
            ucs.printPath(path)
        You, yesterday • Feat: Main added manually input star and goal/end…
            print("Cost = " + str(result.costSoFar))

            display(nodes, adjacencyMatrix, path, startNode)

        else:
            print("File tidak ditemukan. Coba lagi\n")


main()
```

3.4 ucs.py

```python
def uniformCostSearch(nodes, adjacencyMatrix, startNode, goalNode):
    prioQueue = PriorityQueue()
    currentNode = LiveNode(Node(startNode.name, startNode.x, startNode.y, startNode.idx))

    while(currentNode.node.name != goalNode.name):
        # Iterate through all the currentNode neighbors and push them into the prioqueue
        # The prioqueue will check based on Cost of traversal so far
        for i in range(len(adjacencyMatrix[currentNode.node.idx])):
            if(adjacencyMatrix[currentNode.node.idx][i] > 0):
                liveNode = LiveNode(Node(nodes[i].name, nodes[i].x, nodes[i].y, nodes[i].idx))
                liveNode.addCost(adjacencyMatrix[currentNode.node.idx][i])
                liveNode.addPrevNode(currentNode)

                liveNode.defineCompValue(liveNode.costSoFar)    # Using f(n), evaluation with minimum traversal Cost so far

                prioQueue.put(liveNode)

        currentNode = prioQueue.get()

    return currentNode


def reverseString(string):
    reversedString = ""
    for i in range(len(string)-1, -1, -1):
        reversedString += string[i]

    return reversedString
```

```python
def constructPath(liveNode):
    path = []
    prevNodeName = ""
    i = len(liveNode.prevNodes) - 1
    while (i > -1):
        while(liveNode.prevNodes[i] != ' '):
            prevNodeName += liveNode.prevNodes[i]
            i -= 1

        prevNodeName = reverseString(prevNodeName)
        path.append(prevNodeName)
        prevNodeName = ""
        i -= 1

    if(len(path) > 0):
        del path[0]
    path.append(liveNode.node.name)

    return path


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

3.5 star.py

```
def heuristics(akar_node, tujuan_node, heuristic_type):
    if(heuristic_type == '1'):
        akar_node.node.euclideanDistance(tujuan_node)
    else:
        akar_node.node.haversineDistance(tujuan_node)


def astar_search(nodes, adjacencyMatrix, start_node, goal_node, heuristic_type):
    prioQueue = PriorityQueue()
    currentNode = LiveNode(Node(start_node.name, start_node.x, start_node.y, start_node.idx))
    currentNode.node.euclideanDistance(goal_node)

    while(currentNode.node.name != goal_node.name):
        # Iterate through all the currentNode neighbors and push them into the prioqueue
        # The prioqueue will check based on Euclidean/Haversine Distance + Cost of traversal so far (heuristics)
        for i in range(len(adjacencyMatrix[currentNode.node.idx])):
            if(adjacencyMatrix[currentNode.node.idx][i] > 0):
                liveNode = LiveNode(Node(nodes[i].name, nodes[i].x, nodes[i].y, nodes[i].idx))

                heuristics(liveNode, goal_node, heuristic_type)

                liveNode.addCost(adjacencyMatrix[currentNode.node.idx][i])
                liveNode.addPrevNode(currentNode)

                # Using f(n) + h(n)
                # Evaluation with the traversal Cost so far + Euclidean/Haversine Distance of the live node with the goal node
                liveNode.defineCompValue(liveNode.costSoFar + liveNode.node.distance)

                prioQueue.put(liveNode)

        currentNode = prioQueue.get()

    return currentNode
```

3.6 filereader.py

```
def readFile(filename):
    text = ""
    with open(filename) as file:
        for line in file:
            text += line.rstrip()
            text += '\n'

    return text


def ignoreFirstLine(filename):
    # Ignores the first line of text in the text file "filename"
    # Returns the index of the second line of text
    text = readFile(filename)
    i = 0
    while(text[i] != '\n'):
        i += 1

    return i + 1


def getNoOfNodes(filename):
    text = readFile(filename)
    noOfNodes = ""
    i = 0
    while(text[i] != '\n'):
        noOfNodes += text[i]
        i += 1

    return int(noOfNodes)
```

```
def generateNodes(filename):
    x = ""
    y = ""
    nodeName = ""
    nodes = []
    text = readFile(filename)

    i = ignoreFirstLine(filename)
    for j in range(getNoOfNodes(filename)):
        while(text[i] == ' '):
            i += 1

        while(text[i] != ' '):
            nodeName += text[i]
            i += 1

        while(text[i] == ' '):
            i += 1

        while(text[i] != ' '):
            x += text[i]
            i += 1

        while(text[i] == ' '):
            i += 1

        while(text[i] != '\n'):
            y += text[i]
            i += 1

        nodes.append(Node(nodeName, float(x), float(y), j))
        nodeName = ""
        x = ""
        y = ""
        i += 1

    return nodes
```

```
def ignoreNodes(filename):
    # Ignores the node lines in the file text "filename"
    # Returns the first index of the adjacency matrix line in the file text
    text = readFile(filename)
    i = ignoreFirstLine(filename)
    for j in range(getNoOfNodes(filename)):
        while(text[i] != '\n'):
            i += 1
        i += 1
    return i


def generateAdjacencyMatrix(filename):
    adjacencyMatrix = []
    text = readFile(filename)
    elmt = ""

    i = ignoreNodes(filename)
    for j in range(getNoOfNodes(filename)):
        adjacencyMatrix.append([])
        while(text[i] != '\n'):
            while(text[i] == ' '):
                i += 1

            while(text[i] != ' ' and text[i] != '\n'):
                elmt += text[i]
                i += 1

            adjacencyMatrix[j].append(float(elmt))
            elmt = ""

            while(text[i] == ' '):
                i += 1

        i += 1

    return adjacencyMatrix
```

3.7 displaymap.py

```
def fillArrayFromNodeField(nodes, attribute):
    # Returns an array with the nodes attributes
    attributes = []
    for i in range(len(nodes)):
        if(attribute == 'name'):
            attributes.append(nodes[i].name)
        if(attribute == 'x'):
            attributes.append(nodes[i].x)
        if(attribute == 'y'):
            attributes.append(nodes[i].y)

    return attributes
```

```python
def createMap(nodes, adjacencyMatrix, path):
    lon = fillArrayFromNodeField(nodes, 'y')
    lat = fillArrayFromNodeField(nodes, 'x')
    name = fillArrayFromNodeField(nodes, 'name')
    # Make a data frame with dots to show on the map
    data = pd.DataFrame({
    'lon':lon,
    'lat':lat,
    'name':name,
    }, dtype=str)

    m = folium.Map()
    # add marker one by one on the map
    for i in range(0,len(data)):
        folium.Marker(
            location=[data.iloc[i]['lat'], data.iloc[i]['lon']],
            popup=data.iloc[i]['name'],
        ).add_to(m)

    # Add routes
    edgeColor = 'blue'
    for i in range(len(nodes)):
        for j in range(len(nodes)):
            if(adjacencyMatrix[i][j] > 0):
                if(checkIfPath(nodes[i], nodes[j], path)):
                    edgeColor = 'red'

                loc = [(nodes[i].x, nodes[i].y),
                       (nodes[j].x, nodes[j].y)]

                folium.PolyLine(loc,
                                color=edgeColor,
                                opacity=0.8).add_to(m)
                edgeColor = 'blue'

    m.save("map.html")
```

3.8 displaygraph.py

```
def checkIfPath(node1, node2, path):
    if(len(path) == 2):
        if(path[0] == node1.name):
            if(path[1] == node2.name):
                return True
        if(path[1] == node1.name):
            if(path[0] == node2.name):
                return True
    else:
        for i in range(1, len(path)-1):
            if(path[i] == node1.name):
                if(path[i-1] == node2.name or path[i+1] == node2.name):
                    return True

            if(path[i] == node2.name):
                if(path[i-1] == node1.name or path[i+1] == node1.name):
                    return True

    return False
```

```
def displayGraph(nodes, adjacencyMatrix, path):
    G = nx.Graph()
    edgeColor = 'b'

    for i in range(len(nodes)):
        G.add_node(nodes[i].name, pos=(nodes[i].x, nodes[i].y))
        for j in range(len(nodes)):
            if(adjacencyMatrix[i][j] > 0):
                if(checkIfPath(nodes[i], nodes[j], path)):
                    edgeColor = 'r'
                G.add_edge(nodes[i].name, nodes[j].name, color=edgeColor, weight=adjacencyMatrix[i][j])
                edgeColor = 'b'

    pos = nx.get_node_attributes(G,'pos')

    options = {
        "font_size": 24,
        "node_size": 1000,
        "node_color": "skyblue",
        "linewidths": 5,
        "width": 5,
    }
    edges = G.edges()
    colors = [G[u][v]['color'] for u,v in edges]
    weights = [G[u][v]['weight'] for u,v in edges]

    nx.draw_networkx(G, pos, edges, edge_color=colors, **options)

    edge_labels = nx.get_edge_attributes(G, "weight")
    nx.draw_networkx_edge_labels(G, pos, edge_labels)

    # Set margins for the axes so that nodes aren't clipped
    ax = plt.gca()
    ax.margins(0.10)
    plt.axis("off")
    plt.show()
```

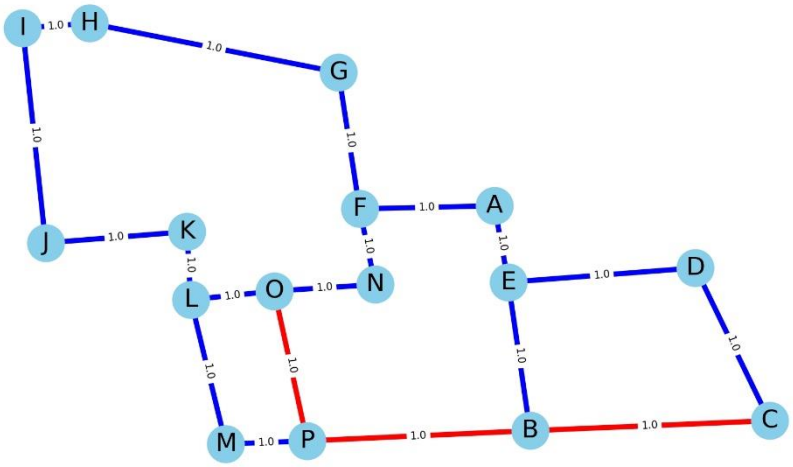
BAB IV

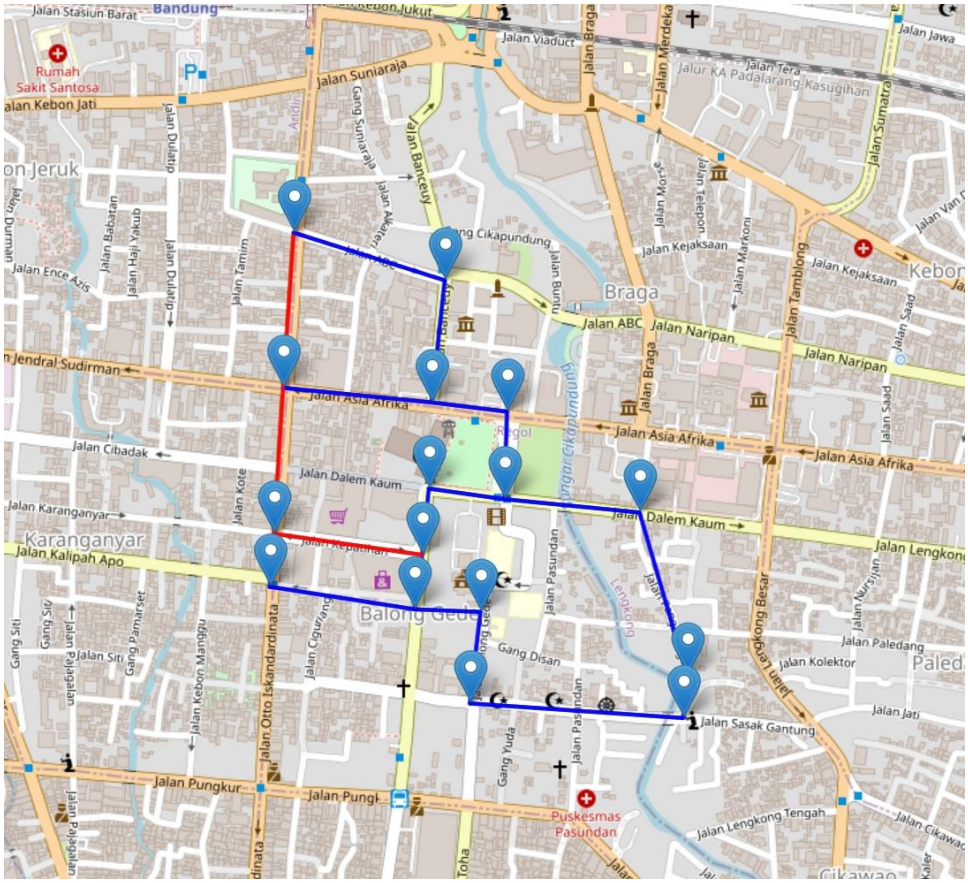
INPUT DAN OUTPUT PROGRAM

4.1 Test pada alun_alun.txt

4.1.1 Dengan Algoritma UCS

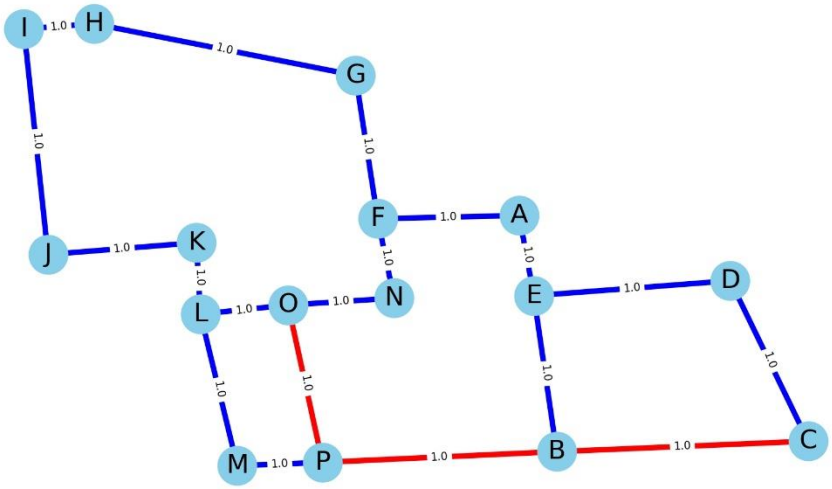
```
Masukkan nama file txt (bukan path & tanpa .txt) : alun_alun
Masukkan start node : C
Masukkan end node : O
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 1
C -> B -> P -> O
Cost = 3.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

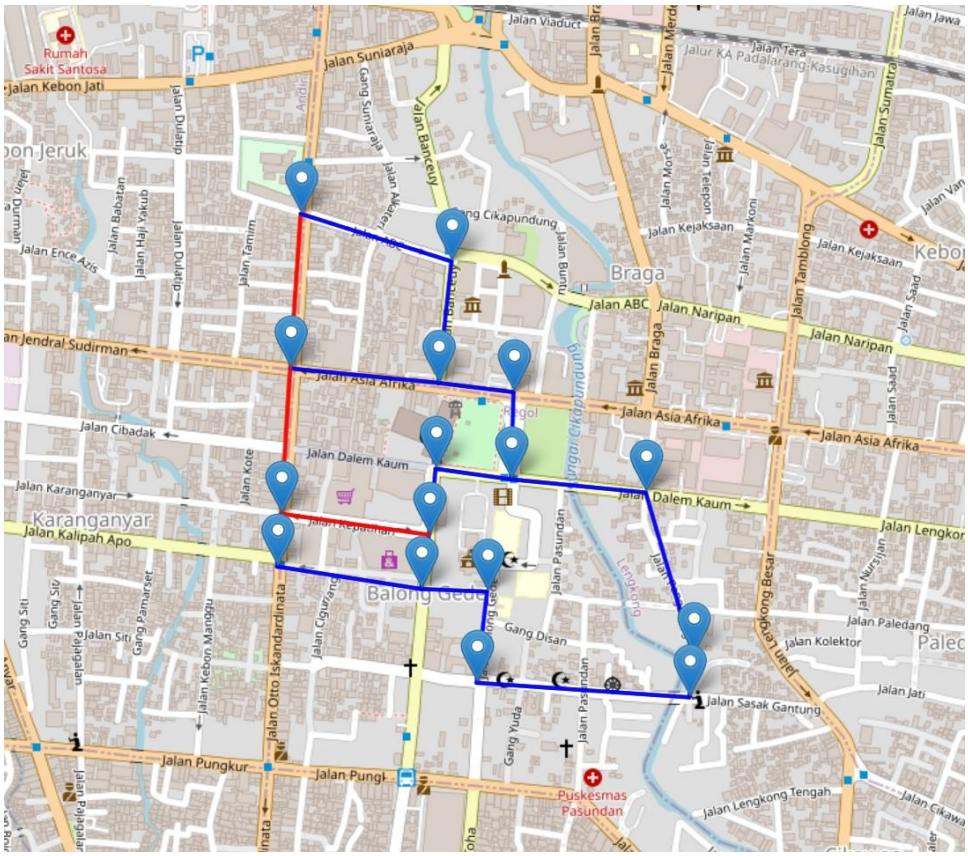




4.1.2 Dengan Algoritma A* (Heuristik Euclidean)

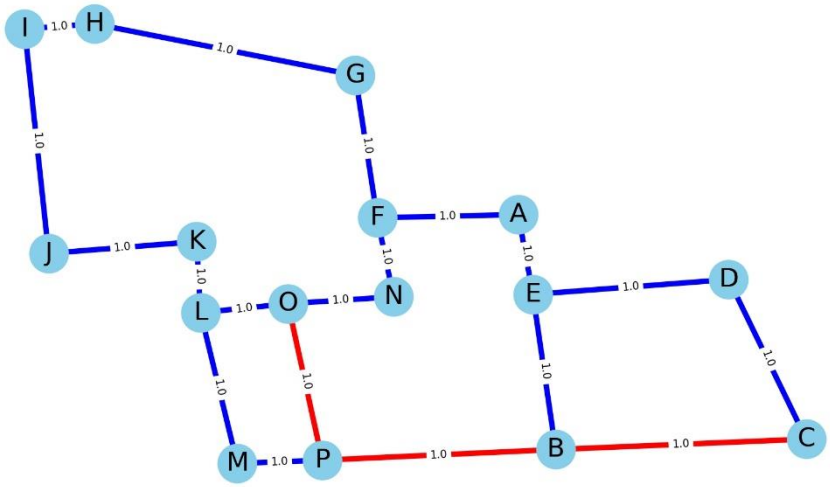
```
Masukkan nama file txt (bukan path & tanpa .txt) : alun_alun
Masukkan start node : C
Masukkan end node : O
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 1
C -> B -> P -> O
Cost = 3.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

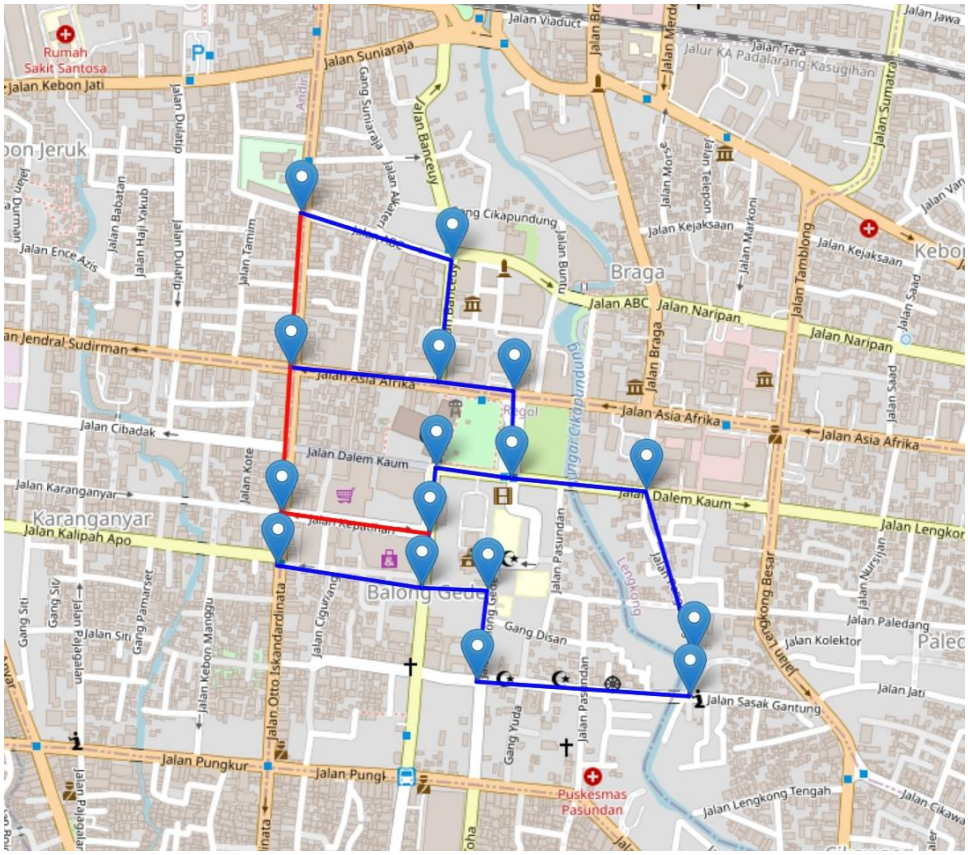

I
H
G
F
A
J
K
L
O
N
E
D
M
P
B
C
1.0


Rumah Sakit Santosa
Jalan Kebon Jati
Jalan Suniaraja
Jalan Viaduct
Jalur KA Padalarang-Kasugihan
Jalan Jawa
Jalan Sumatra
Braga
Jalan ABC
Jalan Naripan
Jalan Asia Afrika
Jalan Dalem Kaum
Jalan Cibadak
Jalan Karanganyar
Karanganyar
Jalan Kalipah Apo
Balong Gede
Gang Disan
Jalan Sasak Gantung
Jalan Pungkur
Puskesmas Pasundan
Jalan Lengkong Tengah
Jalan Kolektor
Jalan Paledang
Jalan Jati

4.1.3 Dengan Algoritma A* (Heuristik Haversine)

```
Masukkan nama file txt (bukan path & tanpa .txt) : alun_alun
Masukkan start node : C
Masukkan end node : O
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 2
C -> B -> P -> O
Cost = 3.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

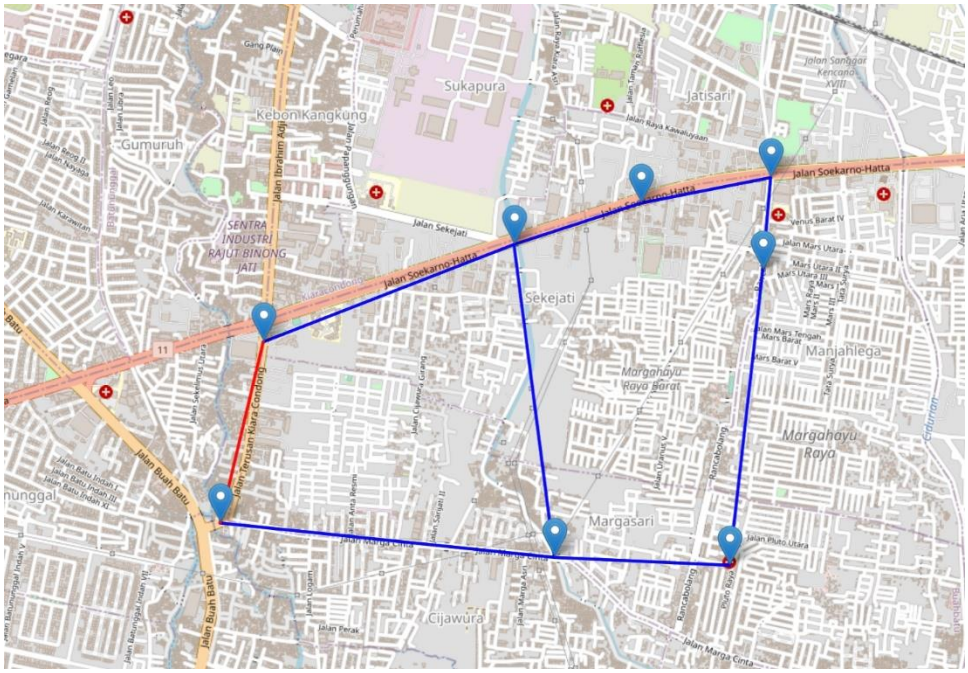




4.2 Test pada buahbatu.txt

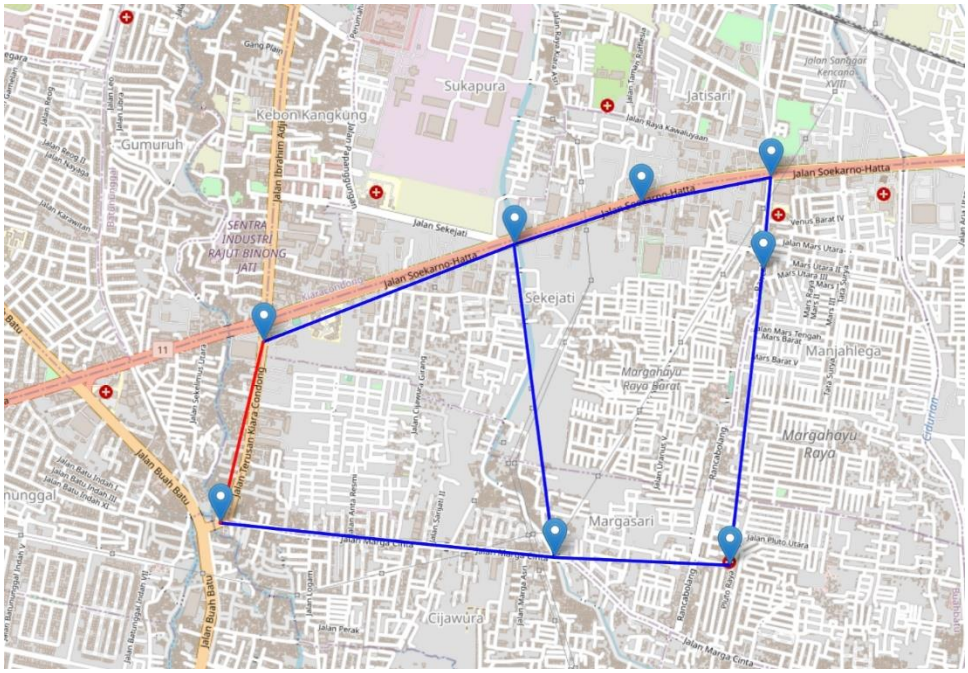
4.2.1 Dengan Algoritma UCS

```
Masukkan nama file txt (bukan path & tanpa .txt) : buahbatu
Masukkan start node : CarrefourKiaracondong
Masukkan end node : MasjidBuahBatu
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 1
CarrefourKiaracondong -> MasjidBuahBatu
Cost = 1.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```



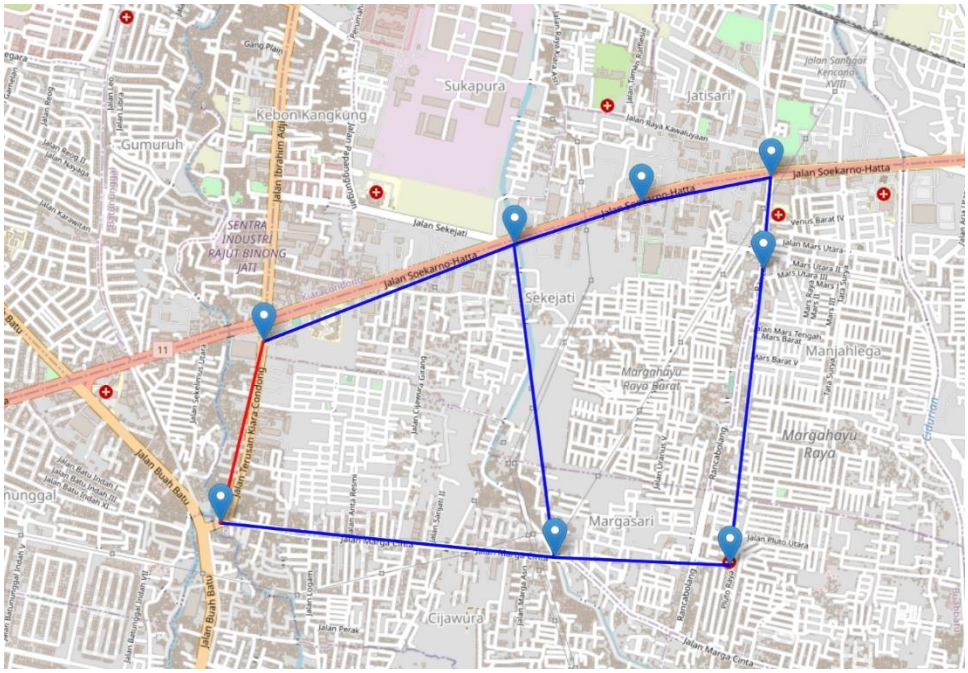
4.2.2 Dengan Algoritma A* (Heuristik Euclidean)

```
Masukkan nama file txt (bukan path & tanpa .txt) : buahbatu
Masukkan start node : CarrefourKiaracondong
Masukkan end node : MasjidBuahBatu
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 1
CarrefourKiaracondong -> MasjidBuahBatu
Cost = 1.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```



4.2.3 Dengan Algoritma A* (Heuristik Haversine)

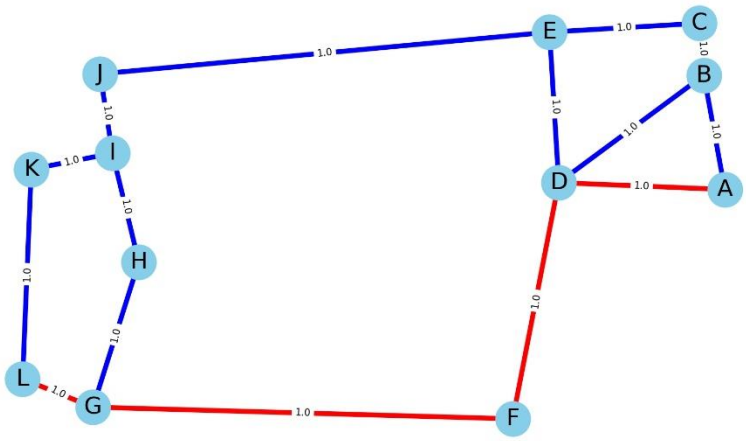
```
Masukkan nama file txt (bukan path & tanpa .txt) : buahbatu
Masukkan start node : CarrefourKiaracondong
Masukkan end node : MasjidBuahBatu
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 2
CarrefourKiaracondong -> MasjidBuahBatu
Cost = 1.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

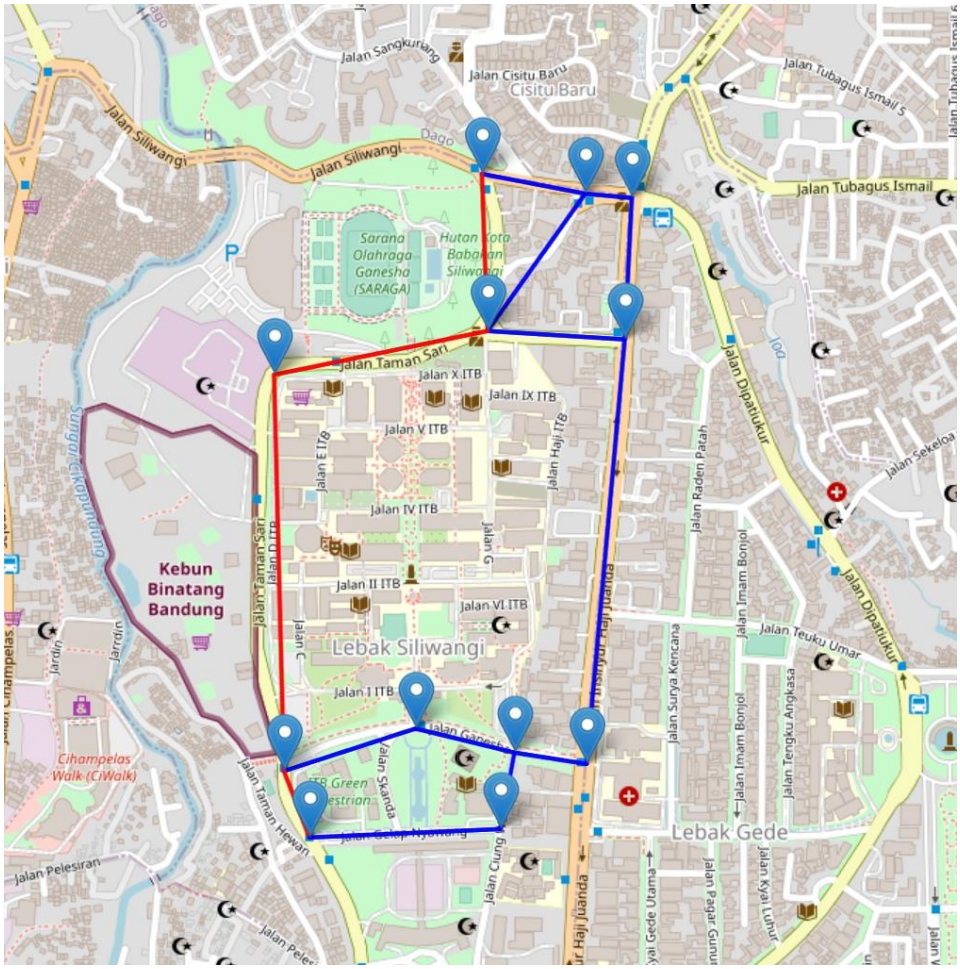


4.3 Test pada itb.txt

4.3.1 Dengan Algoritma UCS

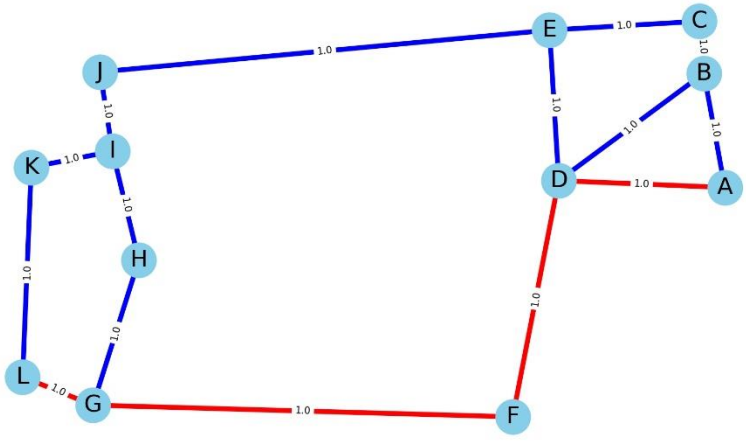
```
Masukkan nama file txt (bukan path & tanpa .txt) : itb
Masukkan start node : A
Masukkan end node : L
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 1
A -> D -> F -> G -> L
Cost = 4.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

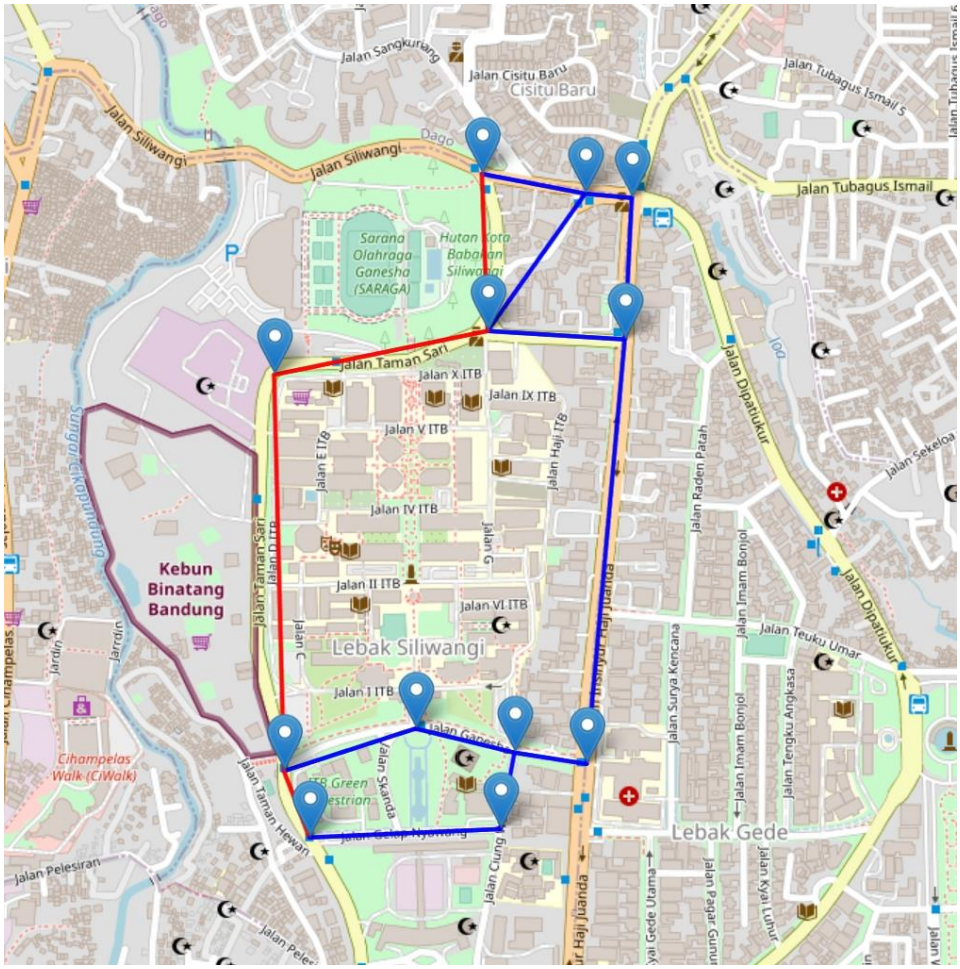

E
C
J
B
I
K
D
A
H
L
G
F
1.0


Jalan Sangkuriang
Jalan Cisitu Baru
Cisitu Baru
Jalan Siliwangi
Dago
Jalan Tubagus Ismail
Sarana Olahraga Ganesha (SARAGA)
Jalan Taman Sari
Jalan X ITB
Jalan IX ITB
Jalan V ITB
Jalan IV ITB
Jalan II ITB
Jalan VI ITB
Jalan I ITB
Jalan Dipatiukur
Jalan Raden Patah
Jalan Imam Bonjol
Jalan Teuku Umar
Jalan Surya Kencana
Jalan Tengku Angkasa
Kebun Binatang Bandung
Lebak Siliwangi
Lebak Gede
Cihampelas Walk (CiWalk)
Jalan Taman Hewan
Jalan Pelesiran
Jalan Ganesha
Jalan Skanda
Jalan Cilaki
Jalan Cikapundung Barat
Jalan Haji Juanda

4.3.2 Dengan Algoritma A* (Heuristik Euclidean)

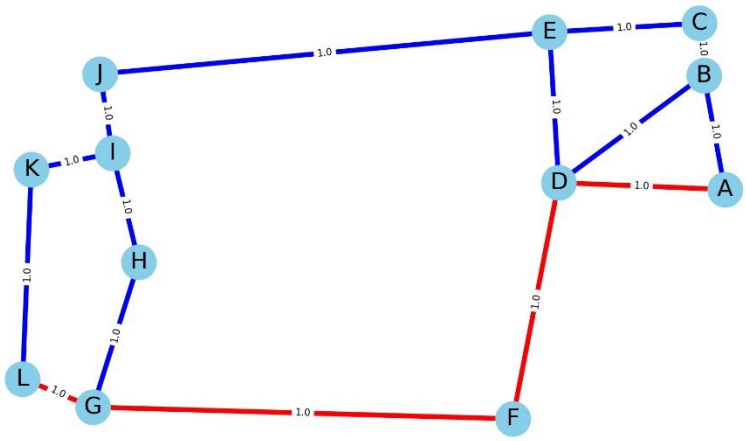
```
Masukkan nama file txt (bukan path & tanpa .txt) : itb
Masukkan start node : A
Masukkan end node : L
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 1
A -> D -> F -> G -> L
Cost = 4.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

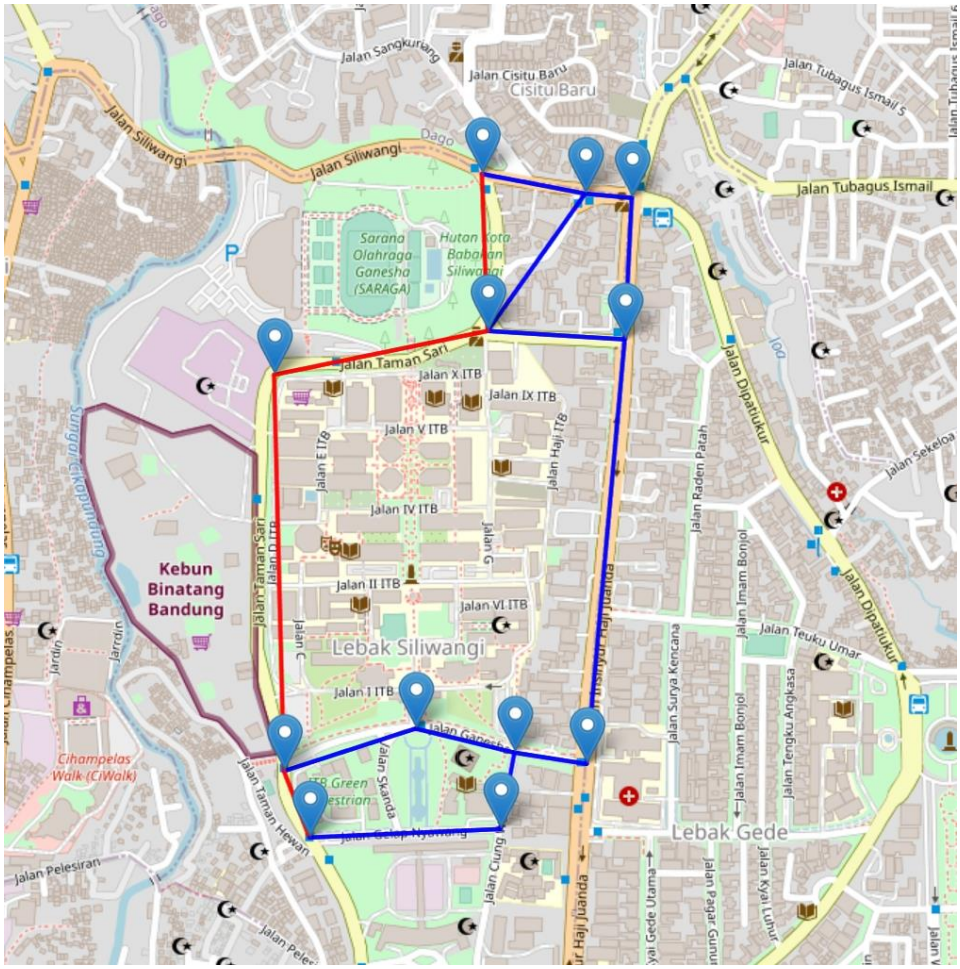




4.3.3 Dengan Algoritma A* (Heuristik Haversine)

```
Masukkan nama file txt (bukan path & tanpa .txt) : itb
Masukkan start node : A
Masukkan end node : L
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 2
A -> D -> F -> G -> L
Cost = 4.0
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 1
```

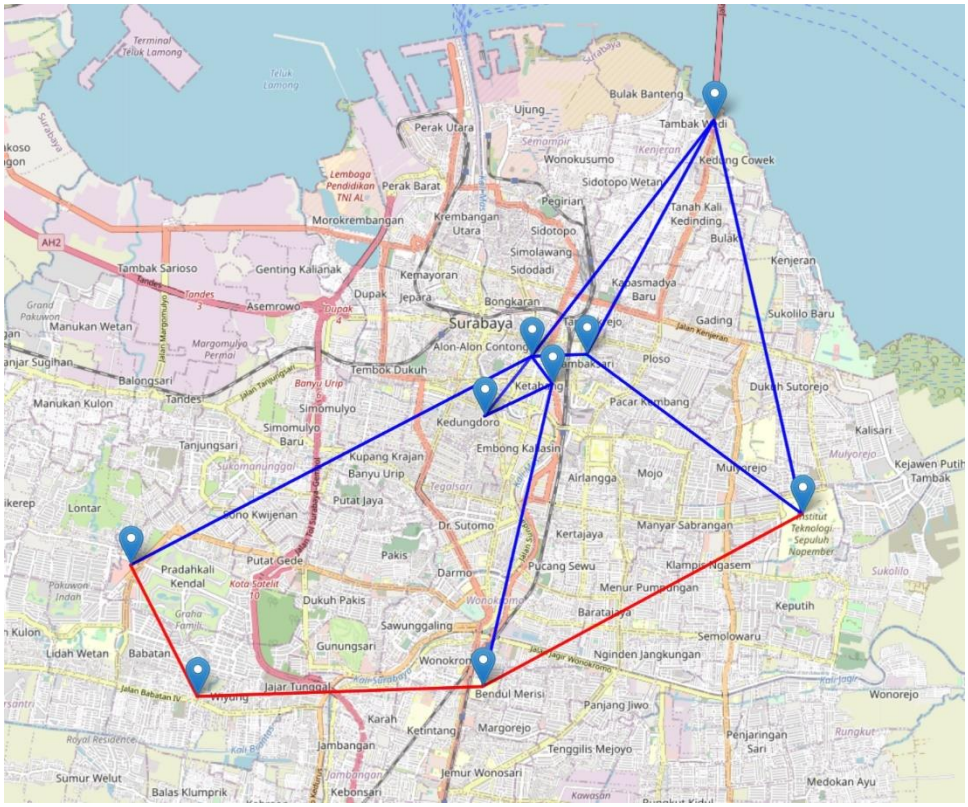

E
C
J
B
I
K
D
A
H
L
G
F
1.0


Jalan Sangkuriang
Jalan Cisitu Baru
Cisitu Baru
Jalan Siliwangi
Dago
Jalan Tubagus Ismail
Sarana Olahraga Ganesha (SARAGA)
Jalan Taman Sari
Jalan X ITB
Jalan IX ITB
Jalan V ITB
Jalan IV ITB
Jalan II ITB
Jalan VI ITB
Jalan I ITB
Jalan Dipatiukur
Jalan Raden Patah
Jalan Imam Bonjol
Jalan Teuku Umar
Jalan Surya Kencana
Jalan Tengku Angkasa
Kebun Binatang Bandung
Lebak Siliwangi
Lebak Gede
Cihampelas Walk (CiWalk)
Jalan Taman Hewan
Jalan Pelesiran
Jalan Ganesha
Jalan Skanda
Jalan Cilaki
Jalan Cikapundung Barat
Jalan Haji Juanda

4.4 Test pada surabaya.txt

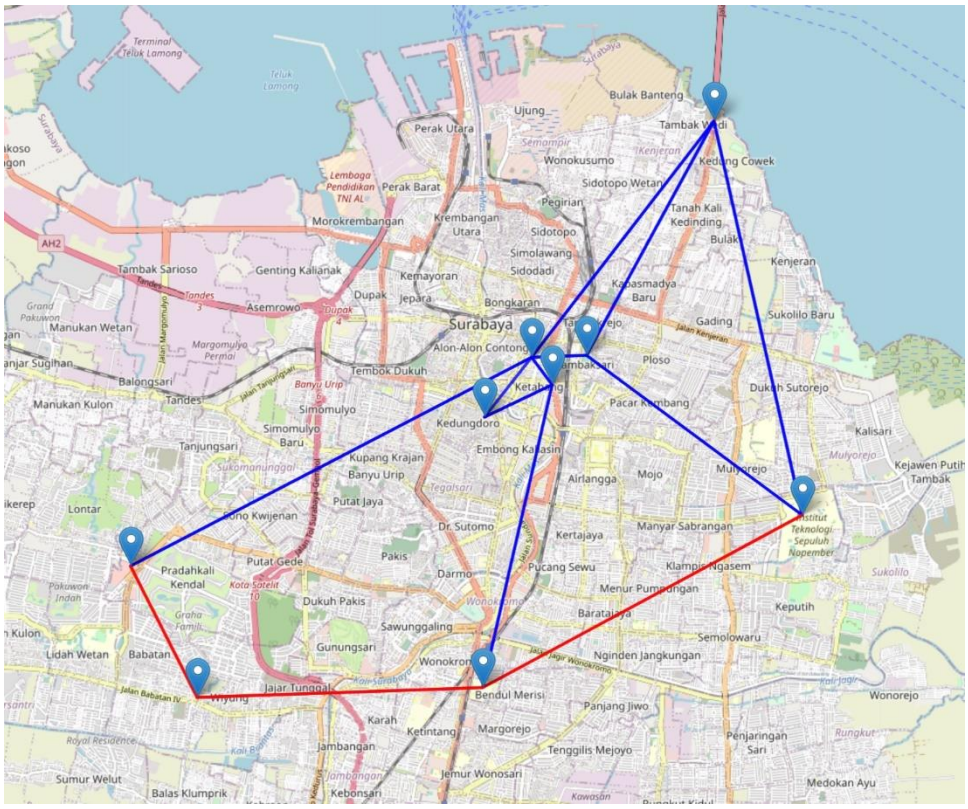
4.4.1 Dengan Algoritma UCS

```
Masukkan nama file txt (bukan path & tanpa .txt) : surabaya
Masukkan start node : PTC
Masukkan end node : ITS
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 1
PTC -> RSWiyungSejahtera -> RSAL -> ITS
Cost = 22.900000000000002
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



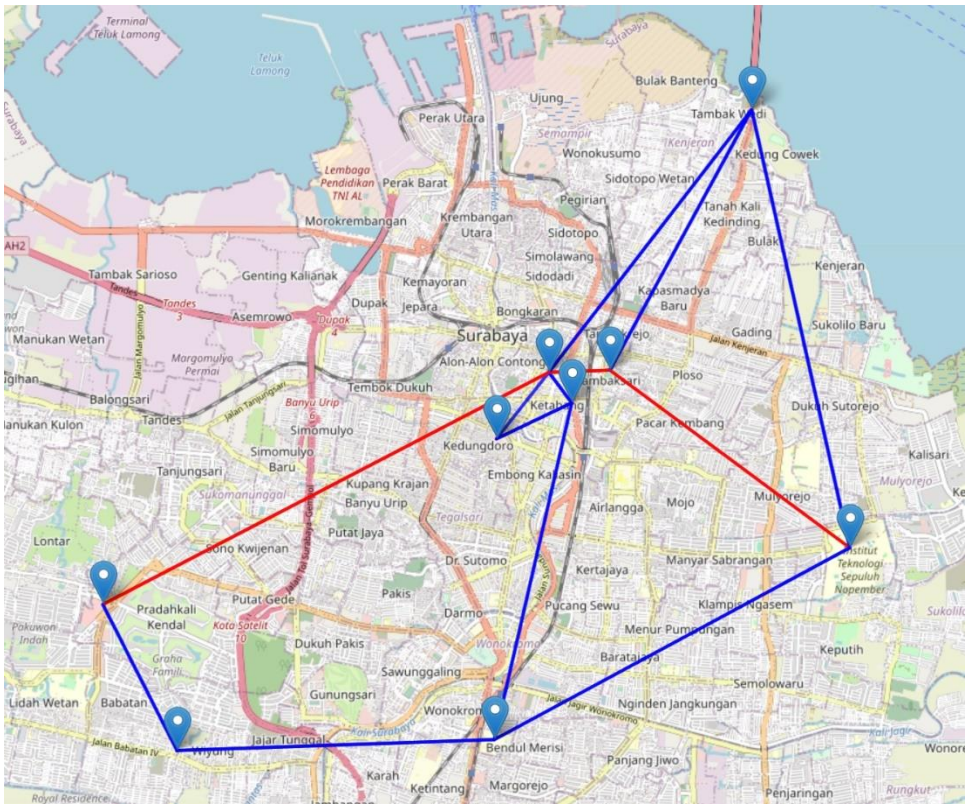
4.4.2 Dengan Algoritma A* (Heuristik Euclidean)

```
Masukkan nama file txt (bukan path & tanpa .txt) : surabaya
Masukkan start node : PTC
Masukkan end node : ITS
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 1
PTC -> RSWiyungSejahtera -> RSAL -> ITS
Cost = 22.900000000000002
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



4.4.3 Dengan Algoritma A* (Heuristik Haversine)

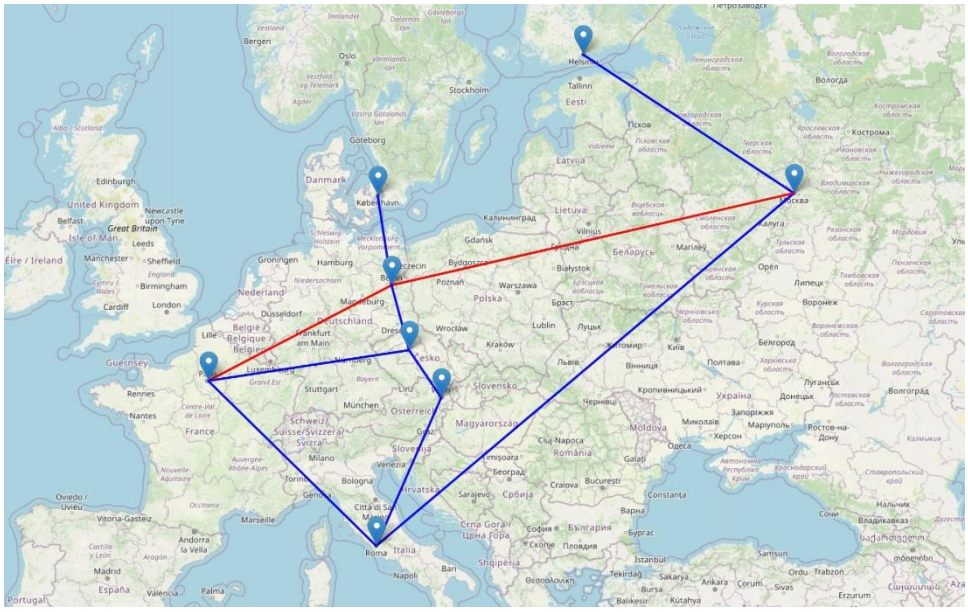
```
Masukkan nama file txt (bukan path & tanpa .txt) : surabaya
Masukkan start node : PTC
Masukkan end node : ITS
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 2
PTC -> MasjidChengHo -> Gelora10November -> ITS
Cost = 23.9
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



4.5 Test pada europecapitals.txt

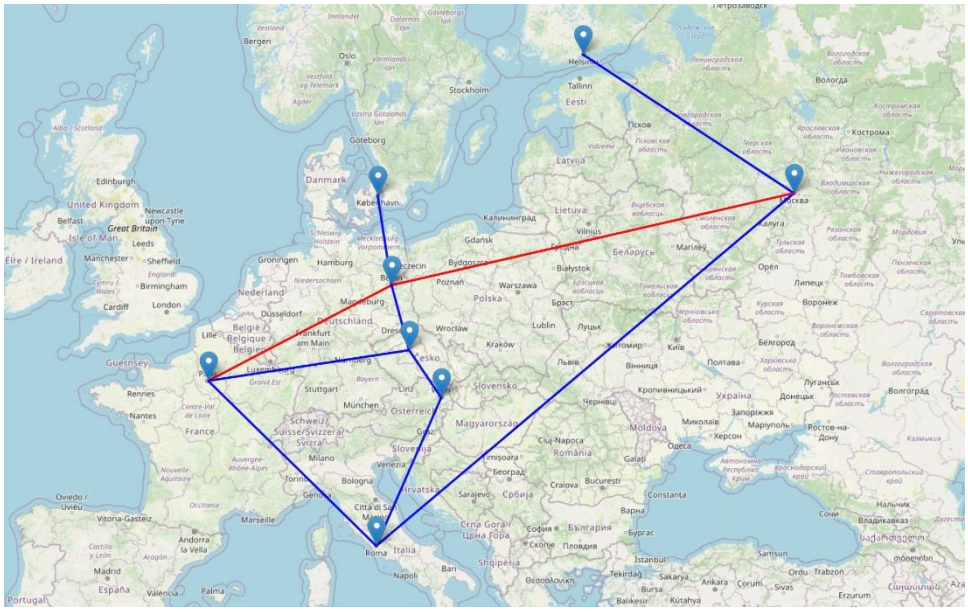
4.4.1 Dengan Algoritma UCS

```
Masukkan nama file txt (bukan path & tanpa .txt) : europecapitals
Masukkan start node : Paris
Masukkan end node : Moscow
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 1
Paris -> Berlin -> Moscow
Cost = 2883.1
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



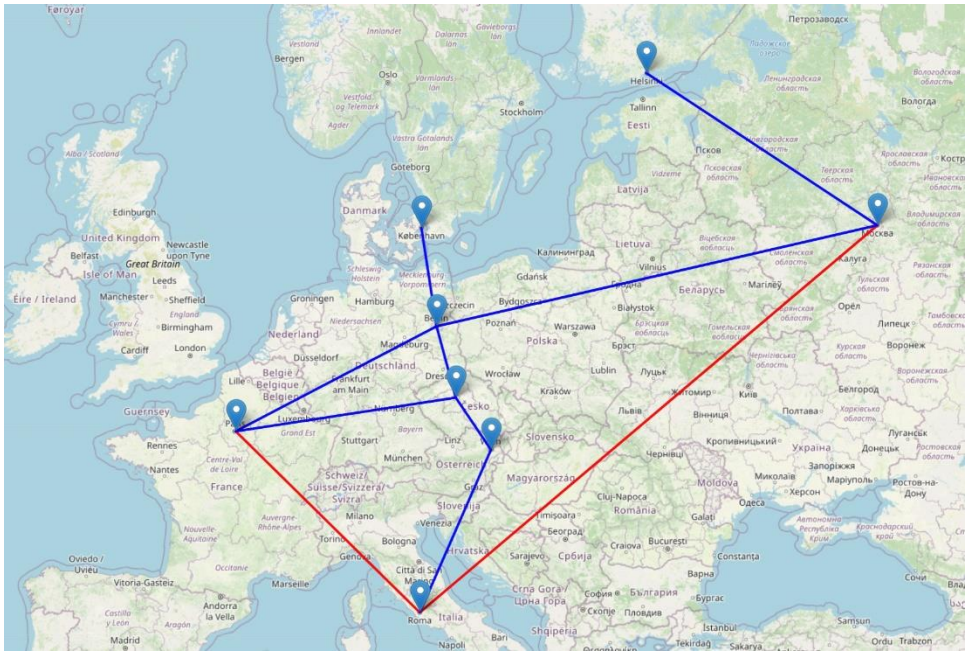
4.4.2 Dengan Algoritma A* (Heuristik Euclidean)

```
Masukkan nama file txt (bukan path & tanpa .txt) : europecapitals
Masukkan start node : Paris
Masukkan end node : Moscow
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 1
Paris -> Berlin -> Moscow
Cost = 2883.1
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



4.4.3 Dengan Algoritma A* (Heuristik Haversine)

```
Masukkan nama file txt (bukan path & tanpa .txt) : europecapitals
Masukkan start node : Paris
Masukkan end node : Moscow
Algoritma :
1. Uniform Cost Search
2. A*
Pilih nomor algoritma : 2
Tipe Heuristik :
1. Euclidean Distance
2. Haversine Distance
Pilih nomor heuristic : 2
Paris -> Rome -> Moscow
Cost = 4458.1
Pilih display dengan :
1. Network Graph
2. Map
Pilih nomor display : 2

Membuat map . . .
Map telah dibuat dengan file map.html pada root
```



KESIMPULAN

Algoritma Uniform Cost Search dan Algoritma A* merupakan algoritma yang relatif mudah untuk dipahami dan diimplementasikan karena kedua algoritma tersebut cukup intuitif. Secara efisiensi program, kedua algoritma tersebut tidak memiliki perbedaan dengan input graf/peta yang kecil. Algoritma UCS dan Algoritma A* dengan heuristik Euclidean Distance yang diimplementasikan pada program menghasilkan rute dengan jarak/cost terkecil, namun seperti yang dapat dilihat pada test case 4 dan 5, Algoritma A* dengan heuristik Haversine Distance masih belum menghasilkan rute dengan jarak/cost terkecil.

LAMPIRAN

Link Repository Github:

https://github.com/rayhanp1402/Tucil3_13521112_13521167

Check List:

1	Program dapat menerima input graf	√
2	Program dapat menghitung lintasan terpendek dengan UCS	√
3	Program dapat menghitung lintasan terpendek dengan A*	√
4	Program dapat menampilkan lintasan terpendek serta jaraknya	√
5	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	Menerima input peta dengan Google Map API : x Menampilkan peta serta lintasannya terpendek pada peta : √

DAFTAR PUSTAKA

Munir, Rinaldi (2022). *Bahan Kuliah IF2211 Strategi Algoritma – Penentuan rute (Route/Path Planning) – Bagian 1*. Diakses pada 6 Februari 2023 pukul 13.17 dari sumber https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

Munir, Rinaldi (2022). *Bahan Kuliah IF2211 Strategi Algoritma – Penentuan rute (Route/Path Planning) – Bagian 2*. Diakses pada 6 Februari 2023 pukul 14.01 dari sumber https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

networkx.org. *networkx documentation*. Diakses pada 10 Februari 23.42 dari sumber https://networkx.org/documentation/stable/reference/index.html

Folium. *Folium 0.14.0 documentation*. Diakses pada 12 Februari 13.19 dari sumber https://python-visualization.github.io/folium/