

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma 2023

Mencari Rute Terpendek dengan Algoritma Uniform Cost Search dan A*

Disusun untuk memenuhi tugas kecil mata kuliah IF2211 Strategi Algoritma pada Semester 2 Tahun Akademik 2022/2023



Disusun oleh:

Kelvin Rayhan Alkarim 13521005

Ditra Rizqa Amadia 13521019

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

BAB I

TEORI DASAR

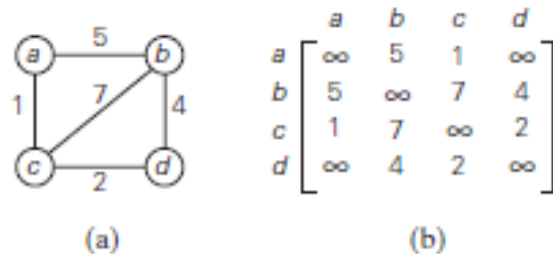
1.1. Perencanaan Rute

Perencanaan rute atau *route planning* adalah proses membuat petunjuk arah mengemudi dari suatu tempat awal menuju tempat tujuan di peta melewati beberapa tempat lainnya dan menyusunnya ke dalam urutan yang paling logis dan praktis. Beberapa algoritma untuk menyelesaikan masalah ini yaitu *Breadth-First Search*, *Depth-First Search*, *Uniform Cost Search*, *A**, dan Dijkstra.

1.2. Graf Berbobot

Suatu graf dapat direpresentasikan sebagai kumpulan titik, beberapa di antaranya menghubungkan oleh suatu garis yang disebut sisi. Graf digunakan untuk banyak aplikasi termasuk transportasi, komunikasi, koneksi ekonomi sosial, jadwal proyek, dan *video game*. Terdapat beberapa masalah yang dapat direpresentasikan oleh graf. Beberapa di antaranya yaitu graph-traversal problems, shortest-path problems, dan topological sorting problems.

Graf berbobot dapat digunakan untuk merepresentasikan suatu peta. Kumpulan titik pada graf dapat merepresentasikan suatu lokasi pada peta. Sisi dapat menghubungkan beberapa titik untuk memberi informasi bahwa suatu titik tersambung dengan titik lainnya. Sedangkan matriks pada graf berbobot yang disebut *weight matrix* atau *cost matrix* merepresentasikan jarak yang harus ditempuh dari dua titik yang terhubung.



Gambar 1 (a) Graf berbobot. (b) Weight matrix

1.3. Algoritma Uniform Cost Search (UCS)

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian jalur terpendek pada graf dengan bobot. Algoritma UCS mencari jalur dengan biaya terendah dari titik awal ke titik akhir.

Prinsip kerja algoritma UCS adalah dengan mempertimbangkan biaya setiap langkah yang diambil untuk mencapai setiap simpul dalam graf, dan memilih jalur dengan biaya total terendah untuk mencapai setiap simpul tersebut. Algoritma UCS menggunakan struktur data antrian prioritas, di mana simpul dengan biaya terkecil diberikan prioritas lebih tinggi.

Berikut adalah langkah-langkah dasar dalam algoritma UCS:

1. Inisialisasi antrian prioritas dengan simpul awal dan nilai biaya nol.
2. Selama antrian prioritas tidak kosong, ambil simpul dengan nilai biaya terendah.
3. Jika simpul tersebut adalah simpul tujuan, kembalikan jalur dari simpul awal ke simpul tujuan.

4. Jika simpul tersebut bukan simpul tujuan, ekspansi simpul tersebut dan tambahkan ke antrian prioritas dengan nilai biaya total yang dihitung dari simpul awal hingga simpul tersebut.
5. Ulangi langkah 2-4 hingga simpul tujuan ditemukan.

Algoritma UCS memiliki kompleksitas waktu yang relatif tinggi, terutama jika graf yang dicari memiliki banyak simpul dan cabang. Oleh karena itu, algoritma UCS biasanya digunakan pada graf dengan ukuran kecil hingga sedang.

1.4. Algoritma A*

Algoritma A* adalah algoritma pencarian rute terpendek dari suatu titik asal menuju titik akhir. Algoritma ini sering digunakan untuk mencari rute terpendek pada suatu peta. A* awalnya didesain untuk memecahkan masalah *graph traversal*. Algoritma ini mencari rute terpendek sehingga rute yang didapat optimal dan lengkap. Algoritma optimal berarti menghasilkan solusi dengan biaya yang minimal sedangkan algoritma lengkap berarti algoritma dapat menemukan semua solusi yang mungkin. Pada implementasinya, A* menggunakan informasi graf dengan graf berbobot.

Tahapan algoritma A* adalah sebagai berikut:

1. Menginisialisasi Open List dan Close List. Open List dan Close List merupakan Array atau List yang menyimpan titik pada graf atau peta
2. Menginisialisasi $f(n)$, $g(n)$, dan $h(n)$. $g(n)$ adalah jarak dari titik awal ke titik n . $h(n)$ adalah estimasi berupa jarak langsung (ditarik dengan garis lurus) dari titik n ke titik tujuan. $f(n)$ adalah jarak dari titik awal menuju titik akhir dibentuk dari $g(n) + h(n)$.
3. Menentukan *expand node* atau titik ekspansi dari Open List. Titik ekspansi didapat dari titik dengan nilai $f(n)$ terkecil pada Open List.
4. Mencari titik yang terhubung dengan titik ekspansi (titik tetangga) dan memasukkannya ke dalam OpenList. Apabila titik sudah berada di dalam Close List, maka titik dikeluarkan dari Close List terlebih dahulu sebelum dimasukkan ke Open List. Nilai $f(n)$ dan $g(n)$ masing-masing titik tetangga diperbaharui sesuai dengan ekspansi node yang sedang diperiksa.
5. Titik ekspansi sudah diperiksa dan dikeluarkan dari Open List lalu dimasukkan ke dalam Close List.
6. Tahapan diulang kembali dari tahap 3 sampai titik ekspansi yang didapat merupakan titik akhir dari permasalahan.
7. Akhirnya, titik yang telah didapat di-*traceback* sehingga didapat rute dari titik awal sampai titik akhir.

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman Python. Struktur dari program ini terbagi menjadi 7 file yaitu main.py, Window.py, Controller.py, Graph.py, Node.py, Astar.py, dan UCS.py.

2.1. File main.py

File ini merupakan driver utama dari program ini, sehingga tidak terdapat fungsi di dalamnya, hanya berisi deklarasi variabel yang akan digunakan.

2.2. Window.py

File ini menyimpan Class Window untuk ditampilkan pada GUI.

Methods	Description
<code>__init__</code>	Konstruktor kelas Window
<code>toggleMenu()</code>	Membuka sidebar
<code>closeMenu()</code>	Menutup sidebar
<code>openMenu()</code>	Memberikan animasi saat membuka dan menutup sidebar
<code>openFileDialog</code>	Membuka dialog file untuk memilih file yang akan diimport
<code>createGMPlot</code>	Membuat Google Map Plot dan disimpan pada file temp.html
<code>createNetworkX()</code>	Membuat sebuah graf untuk ditampilkan pada plot
<code>pairRouteEdge()</code>	Menerima List rute dan mengembalikan List berisi sepasang node yang bertetanggan untuk digunakan pada plot graf
<code>searchRoute()</code>	Mencari rute dengan pilihan algoritma UCS maupun A*

2.3. Controller.py

File ini menyimpan Class Controller yang akan digunakan untuk mengatur data dan algoritma pada program melalui GUI. Controller memiliki beberapa atribut yaitu graph dan solution.

Methods	Description
---------	-------------

<code>__init__</code>	Konstruktor kelas Controller
<code>getSolution()</code>	Mengembalikan atribut solusi
<code>getSolutionDistance()</code>	Mengembalikan distance solusi
<code>getSolutionRoute()</code>	Mengembalikan rute solusi berupa String
<code>getSolutionList()</code>	Mengembalikan rute solusi dalam bentuk List of Integer
<code>getGraph()</code>	Mengembalikan Graph
<code>isNodeValid(nodeId)</code>	Memeriksa apakah node ada pada graf. Mengembalikan True apabila node ada pada graf dan False apabila node tidak ada pada graf
<code>importFile(filePath, fileName)</code>	Membuat graf dan disimpan sebagai atribut
<code>search(isUCS, startNodeId, endNodeId)</code>	Mencari rute terpendek dari startNodeId sampai endNodeId dengan pilihan algoritma UCS maupun A*

2.4. Graph.py

File ini menyimpan Class Graph untuk merepresentasikan graf suatu data. Graph memiliki beberapa attribute yaitu filePath, fileName, isWCoordinate, adjMatrix, dan nodeList.

Methods	Description
<code>__init__</code>	Konstruktor kelas Graph
<code>getAdjMatrix()</code>	Mengembalikan Adjacent Matrix Graph
<code>getNodeList()</code>	Mengembalikan List of Node pada Graph
<code>getIsWCoordinate()</code>	Mengembalikan True apabila node pada graf memiliki koordinat di dunia nyata. Mengembalikan False apabila node tidak memiliki koordinat
<code>validateFileFormat()</code>	Memeriksa format file yang dimasukan oleh pengguna
<code>buildNodes()</code>	Membuat seluruh node yang ada pada file input

<code>buildAdjMatrix()</code>	Membuat Adjacent Matrix Graph berdasarkan file input
<code>Build()</code>	Membangun graf dengan memanggil <code>validateFileFormat()</code> , <code>buildNodes()</code> , dan <code>buildAdjMatrix()</code>

2.5. Node.py

File ini menyimpan Class Node untuk merepresentasikan node pada suatu graf. Node memiliki beberapa atribut yaitu x, y, id, dan parent.

Methods	Description
<code>__init__</code>	Konstruktor kelas Node
<code>getId()</code>	Mengembalikan id Node
<code>setParent()</code>	Mengubah nilai parent node
<code>getParent()</code>	Mengembalikan parent node
<code>getX()</code>	Mengembalikan posisi x node apabila node memiliki koordinat
<code>getY()</code>	Mengembalikan posisi y node apabila node memiliki koordinat

2.6. AStar.py

File ini menyimpan Class Algoritma A*. Class ini memiliki beberapa atribut yaitu `openList`, `closedList`, `solution`, `fScore`, `gScore`, `hScore`, `last`, dan `wCoordinate`

Methods	Description
<code>__init__</code>	Konstruktor kelas AStar
<code>getSolution()</code>	Mengembalikan solusi
<code>initialState()</code>	Memberikan nilai awal pada <code>fScore</code> , <code>gScore</code> dan <code>hScore</code>
<code>updateStartingNode(startingNode)</code>	Memperbaharui nilai <code>fScore</code> , <code>gScore</code> , dan <code>hScore</code> starting Node
<code>determineExpandNode()</code>	Mencari node selanjutnya dari expand node saat ini berdasarkan <code>gScore</code> yang paling kecil

<code>updateNeighbourNode(graph, expandNode)</code>	Memperbaharui fScore, gScore, dan hScore node tetangga dari expand node
<code>reconstructPath(startingNode, destinationNode)</code>	Membentuk rute dari atribut last setiap node
<code>findShortestPath(graph, startingNode, destinationNode)</code>	Mencari rute optimal dari starting node menuju destination node dengan bantuan fungsi <code>initiateState()</code> , <code>updateStartingNode()</code> , <code>determineExpandNode()</code> , <code>updateNeighbourNode()</code> , dan <code>reconstructPath()</code>

2.7. UCS.py

File ini menyimpan Class Algoritma UCS. Class ini memiliki beberapa atribut yaitu `visited`, `queue`, dan `solution`.

Methods	Description
<code>__init__</code>	Konstruktor kelas AStar
<code>getSolution()</code>	Mengembalikan solusi
<code>search(graph, start, goal)</code>	Mencari rute optimal dari start menuju goal

2.8. Library

Terdapat juga beberapa library yang digunakan untuk program ini, antara lain :

- `os`
- `sys`
- `math`
- `gmpplot`
- `PyQt5`
- `matplotlib`
- `networkx`
- `scipy`

BAB III

SOURCE CODE PROGRAM

3.1. Repository Program

Repository program dapat diakses melalui tautan GitHub berikut :
https://github.com/ditramadia/TuciI3_13521005_13521019.git

3.2. Source Code Program

3.1.1. main.py

```
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import *
import Window

app = QApplication(sys.argv)
window = Window.MainWindow()
widget = QtWidgets.QStackedWidget()
widget.addWidget(window)
widget.setWindowTitle("Shortest Route Finder")
widget.setFixedHeight(720)
widget.setFixedWidth(1280)
widget.show()
sys.exit(app.exec_())
```

3.1.2. Window.py

```
import os
from PyQt5.uic import loadUi
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import QWebEngineView
from PyQt5 import QtCore
from PyQt5.QtCore import *
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
import matplotlib.figure as Figure
import networkx as nx
import gmplot
import Controller

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
```



```

loadUi('src/uibuilder/Main.ui', self)
self.pushButton.clicked.connect(self.toggleMenu)
self.search_btn.clicked.connect(self.searchRoute)
self.import_btn.clicked.connect(self.openFileDialog)

self.web_view = QWebView(self.plot)
self.web_view.setFixedWidth(1280)
self.web_view.setFixedHeight(720)
self.plot.hide()

widget_size = self.plot.size()
self.figure = Figure.Figure(figsize=(widget_size.width(),
widget_size.height()))
self.figure.set_size_inches(widget_size.width()/100,
widget_size.height()/120)
self.canvas = FigureCanvas(self.figure)
self.canvas.setParent(self.plot)
self.graph = self.figure.add_subplot(111)
self.graph.set_axis_off()
self.canvas.hide()

self.solution.hide()
self.popup_container.hide()

self.__controller = Controller.Controller()

def toggleMenu(self):
    if self.sidebar_container.pos().x() == 0:
        self.closeMenu()
    else:
        self.openMenu()

def closeMenu(self):
    self.animation = QPropertyAnimation(self.sidebar_container, b"pos")
    self.animation.setEasingCurve(QEasingCurve.InOutCubic)
    self.animation.setStartValue(QPoint(0, 0))
    self.animation.setEndValue(QPoint(-321, 0))
    self.animation.setDuration(200)
    self.animation.start()

def openMenu(self):
    self.animation = QPropertyAnimation(self.sidebar_container, b"pos")
    self.animation.setEasingCurve(QEasingCurve.InOutCubic)
    self.animation.setStartValue(QPoint(-321, 0))
    self.animation.setEndValue(QPoint(0, 0))
    self.animation.setDuration(200)
    self.animation.start()

```

```

def openFileDialog(self):
    filePath, _ = QFileDialog.getOpenFileName(self, 'Open file', '',
'Text files (*.txt)')
    fileName = os.path.basename(filePath)
    if filePath:
        try:
            self.__controller.importFile(filePath, fileName)
            self.solution.hide()
            self.file_label.setText(fileName)
            self.popup_container.hide()
            self.plot.hide()

        except Exception as err:
            self.popup_value.setText(f"Input file error: {err.args[0]}")
            self.popup_container.show()

def createGMPLOT(self):
    gmap = gmpplot.GoogleMapPlotter(-6.901837, 107.601241, 13)

    for node in self.__controller.getGraph().getNodeList():
        for node2 in self.__controller.getGraph().getNodeList():
            if self.__controller.getGraph().getAdjMatrix()[node.getId()
- 1][node2.getId() - 1] > 0:
                lats = [node.getX(), node2.getX()]
                lngs = [node.getY(), node2.getY()]
                gmap.scatter(lats, lngs, '#FFA54F', size=10,
marker=False)

                gmap.plot(lats, lngs, '#FFA54F', edge_width=2.5)

    latsSolution = []
    lngsSolution = []
    for node in self.__controller.getSolution()["path"]:
        latsSolution.append(self.__controller.getGraph().getNodeList()[n
ode - 1].getX())
        lngsSolution.append(self.__controller.getGraph().getNodeList()[n
ode - 1].getY())

    gmap.scatter(latsSolution, lngsSolution, '#63B8FF', size=10,
marker=False)
    gmap.plot(latsSolution, lngsSolution, '#63B8FF', edge_width=3.5)
    gmap.draw(f"./test/temp.html")

def createNetworkX(self):
    g = nx.DiGraph()
    nodeList = self.__controller.getGraph().getNodeList()
    node1Idx = 0
    for nodeRow in self.__controller.getGraph().getAdjMatrix():
        node2Idx = 0

```

```

        for weight in nodeRow:
            if weight > 0:
                g.add_edge(nodeList[node1Idx].getId(),
nodeList[node2Idx].getId(), weight=weight)
                node2Idx += 1
            node1Idx += 1
        return g

    def pairRouteEdge(self):
        pair = []
        nodeIdx = 0
        for node in self.__controller.getSolutionRouteList():
            if nodeIdx == len(self.__controller.getSolutionRouteList()) - 1:
                break
            pair.append((int(node),
int(self.__controller.getSolutionRouteList()[nodeIdx + 1])))
            nodeIdx += 1
        return pair

    def searchRoute(self):
        try:
            if self.file_label.text() == "No file":
                raise Exception("No input file")
            if self.start_input.text() == "":
                raise Exception("Start node is missing")
            if self.end_input.text() == "":
                raise Exception("End node is missing")
            if not self.start_input.text().isdigit() or not
self.__controller.isNodeValid(int(self.start_input.text())):
                raise Exception("Start node does not exist")
            if not self.end_input.text().isdigit() or not
self.__controller.isNodeValid(int(self.end_input.text())):
                raise Exception("End node does not exist")
            if not self.radioButton.isChecked() and not
self.radioButton_2.isChecked():
                raise Exception("No algorithm selected")

            if self.radioButton.isChecked():
                self.solution_label.setText("Shortest Route (UCS)")
                self.__controller.search(True, int(self.start_input.text()),
int(self.end_input.text()))
            else:
                self.solution_label.setText("Shortest Route (A*)")
                self.__controller.search(False,
int(self.start_input.text()), int(self.end_input.text()))

            if self.__controller.getGraph().getIsWCoordinate():
                self.canvas.hide()

```

```

        self.createGMPLOT()
        self.web_view.setUrl(QtCore.QUrl.fromLocalFile(os.path.abspath(
th("test/temp.html"))))
        self.plot.show()
    else:
        self.plot.show()
        network = self.createNetworkX()
        pos = nx.kamada_kawai_layout(network)
        pair = self.pairRouteEdge()
        print(pair)
        nx.draw(network, pos, node_size=500, node_color='#FFA54F',
ax=self.graph, font_size=10, font_color="black", font_weight="bold",
with_labels=True)
        nx.draw_networkx_edges(network, pos, edgelist=pair,
edge_color='#63B8FF', width=3, ax=self.graph)
        self.canvas.show()

        self.distance_value.setText(str(self.__controller.getSolutionDis
tance()))
        self.route_value.setText(self.__controller.getSolutionRoute())
        self.solution.show()
        self.toggleMenu()

    except Exception as err:
        self.popup_value.setText(f"Input file error: {err.args[0]}")
        self.popup_container.show()

```

3.1.3. Controller.py

```

import Graph
import UCS
import AStar

class Controller:
    # == CONSTRUCTOR
    =====
    def __init__(self):
        self.__graph = None
        self.__solution = None

    # == GETTER SETTER
    =====
    def getSolution(self):
        return self.__solution

    def getSolutionDistance(self):

```

```

        return self.__solution["distance"]

    def getSolutionRoute(self):
        route = ""
        i = 0
        for node in self.__solution["path"]:
            if i == len(self.__solution["path"]) - 1:
                route += str(node)
                break
            route += str(node) + " - "
            i += 1
        return route

    def getSolutionRouteList(self):
        return self.__solution["path"]

    def getGraph(self):
        return self.__graph

# === VALIDATOR
=====

    def isNodeValid(self, nodeId):
        valid = False
        for node in self.__graph.getNodeList():
            if nodeId == node.getId():
                valid = True
        return valid

# === IO
=====

    def importFile(self, filePath, fileName):
        self.__graph = Graph.Graph(filePath, fileName)
        self.__graph.build()

    def search(self, isUCS, startNodeId, endNodeId):
        if isUCS:
            ucsAlgorithm = UCS.UCS()
            ucsAlgorithm.search(self.__graph, startNodeId, endNodeId)
            self.__solution = ucsAlgorithm.getSolution()
        else:
            aStarAlgorithm = AStar.AStar()
            aStarAlgorithm.findShortestPath(self.__graph, startNodeId,
endNodeId)
            self.__solution = aStarAlgorithm.getSolution()

```

3.1.4. Graph.py

```
import math
import Node

class Graph:
    # == CONSTRUCTOR
    =====
    def __init__(self, filePath, fileName):
        self.__filePath = filePath
        self.__fileName = fileName
        self.__isWCoordinate = False
        self.__adjMatrix = []
        self.__nodeList = []

    # == GETTER SETTER
    =====
    def getAdjMatrix(self):
        return self.__adjMatrix

    def getNodeList(self):
        return self.__nodeList

    def getIsWCoordinate(self):
        return self.__isWCoordinate

    # == INITIALIZER
    =====
    def validateFileFormat(self):

        file = open(self.__filePath)
        charMatrix = []
        for line in file:
            charMatrix.append(line.replace("\n", "").split())
        file.close()

        if len(charMatrix) == len(charMatrix[0]):
            for charList in charMatrix:
                if len(charList) != len(charMatrix):
                    raise Exception("The number of rows and columns does not
match")

                for char in charList:
                    try:
                        float(char)
                    except:
                        raise Exception("All characters must be a numerical
value")

            self.__isWCoordinate = False
```

```

        elif len(charMatrix) + 2 == len(charMatrix[0]):
            for charList in charMatrix:
                if len(charList) != 2 + len(charMatrix):
                    raise Exception("The number of rows and columns does not
match")

                for char in charList:
                    try:
                        float(char)
                    except:
                        raise Exception("All characters must be a numerical
value")

            self.__isWCoordinate = True
        else:
            raise Exception("The number of rows and columns does not match")

    def buildNodes(self):
        self.__nodeList = []

        file = open(self.__filePath, "r")
        nodeId = 1
        for line in file:
            charList = line.replace("\n", "").split()
            if self.__isWCoordinate:
                newNode = Node.Node(nodeId, float(charList[len(charList) -
2]), float(charList[len(charList) - 1]))
            else:
                newNode = Node.Node(nodeId)
            self.__nodeList.append(newNode)
            nodeId += 1
        file.close()

    def buildAdjMatrix(self):
        self.__adjMatrix = []

        file = open(self.__filePath, "r")
        i = 0
        for line in file:
            charList = line.replace("\n", "").split()
            weightList = []

            if self.__isWCoordinate:
                for j in range(0, len(charList) - 2):
                    weightList.append(math.sqrt(((self.__nodeList[i].getX()
- self.__nodeList[j].getX()) ** 2) + ((self.__nodeList[i].getY() -
self.__nodeList[j].getY()) ** 2)) * float(charList[j]))
            else:
                for element in charList:
                    weightList.append(float(element))

```

```

        self.__adjMatrix.append(weightList)
        i += 1
    file.close()

def build(self):
    self.validateFileFormat()
    self.buildNodes()
    self.buildAdjMatrix()

```

3.1.5. Node.py

```

class Node:
    # === CONSTRUCTOR
    =====

    def __init__(self, id, x=None, y=None):
        self.x = x
        self.y = y
        self.id = id
        self.parent = None

    # === GETTER SETTER
    =====

    def getId(self):
        return self.id

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

    def getParent(self):
        return self.parent

    def getX(self):
        return self.x

    def getY(self):
        return self.y

```

3.1.6. UCS.py

```

import Graph as gp
class UCS:
    # === CONSTRUCTOR
    =====

```



```

def __init__(self):
    self.visited = []
    self.queue = []
    self.solution = {
        "path" : [],
        "distance" : None
    }

# == GETTER SETTER
=====

def getSolution(self):
    return self.solution

# == METHODS
=====

def search(self, graph, start, goal):
    listnode = graph.getNodeList()
    matrix = graph.getAdjMatrix()

    self.queue.append([start, 0])

    while len(self.queue) > 0:
        current = self.queue.pop(0)
        self.visited.append(current[0])

        if current[0] == goal:
            node = listnode[goal - 1]
            while node != None:
                self.solution["path"].append(node.getId())

                if (node.getId() == start):
                    break
                node = listnode[node.getParent() - 1]
            self.solution["distance"] = current[1]
            self.solution["path"].reverse()
            break

        for i in range(len(matrix[current[0] - 1])):
            if matrix[current[0] - 1][i] != 0 and (i + 1) not in
self.visited:
                listnode[i].setParent(current[0])
                self.queue.append([i + 1, current[1] +
matrix[current[0] - 1][i]])

    self.queue.sort(key=lambda x: x[1])

```

3.1.7. AStar.py

```
import math

class AStar:
    # == CONSTRUCTOR
    =====

    def __init__(self, wCoordinate = False):
        self.openList = set([])
        self.closedList = set([])
        self.solution = {
            "path" : [],
            "distance" : None
        }
        self.fScore = {}
        self.gScore = {}
        self.hScore = {}
        self.last = {}
        self.wCoordinate = wCoordinate

    # == GETTER SETTER
    =====

    def getSolution(self):
        return self.solution

    # == UTILITY
    =====

    # Initialize score and last with default value
    def initialState(self, graph, destinationNode):
        for node in graph.getNodeList():
            if self.wCoordinate:
                self.hScore[int(node.getId())] =
math.sqrt((((graph.getNodeList()[destinationNode - 1].getX() - node.getX())
** 2) + ((graph.getNodeList()[destinationNode - 1].getY() - node.getY()) **
2))
            else:
                self.hScore[int(node.getId())] = float(9999)
                self.gScore[int(node.getId())] = float("inf")
                self.fScore[int(node.getId())] = float("inf")
                self.last[int(node.getId())] = None

    # Update starting node score
    def updateStartingNode(self, startingNode):
        self.openList.add(startingNode)
        self.gScore[startingNode] = 0
        self.fScore[startingNode] = self.gScore[startingNode] +
self.hScore[startingNode]
```

```

        self.last[startingNode] = startingNode

    # Determine node to expand based on lowest fScore
    def determineExpandNode(self):
        expandNode = None
        for comparingNode in self.openList:
            if expandNode == None or self.gScore[comparingNode] +
self.hScore[comparingNode] < self.gScore[expandNode] +
self.hScore[expandNode]:
                expandNode = comparingNode
        return expandNode

    # Update neighbouring node of expand node and update its score
    def updateNeighbourNode(self, graph, expandNode):
        comparingNode = 1
        for weight in graph.getAdjMatrix()[expandNode - 1]:
            if weight > 0 and weight != float("inf"):
                if comparingNode not in self.openList and comparingNode not
in self.closedList:
                    self.openList.add(comparingNode)
                    self.last[comparingNode] = expandNode
                    self.gScore[comparingNode] = self.gScore[expandNode] +
weight
                    self.fScore[comparingNode] = self.gScore[comparingNode]
+ self.hScore[comparingNode]
                else:
                    if self.gScore[comparingNode] > self.gScore[expandNode]
+ weight:
                        self.gScore[comparingNode] = self.gScore[expandNode]
+ weight
                        self.last[comparingNode] = expandNode
                        if comparingNode in self.closedList:
                            self.closedList.remove(comparingNode)
                            self.openList.add(comparingNode)
                comparingNode += 1

    # Reconstruct solution path
    def reconstructPath(self, startingNode, destinationNode):
        self.solution["path"] = [destinationNode]

        while self.solution["path"][0] != startingNode:
            self.solution["path"].insert(0,
self.last[self.solution["path"][0]])

        self.solution["distance"] = self.gScore[destinationNode]

    # == METHODS
=====

```

```
def findShortestPath(self, graph, startingNode, destinationNode):
    self.initialState(graph, destinationNode)
    self.updateStartingNode(startingNode)
    while True:
        expandNode = self.determineExpandNode()
        if expandNode == destinationNode:
            self.reconstructPath(startingNode, destinationNode)
            return
        self.updateNeighbourNode(graph, expandNode)
        self.openList.remove(expandNode)
        self.closedList.add(expandNode)
```

BAB IV

MASUKAN DAN LUARAN PROGRAM

Graph 1

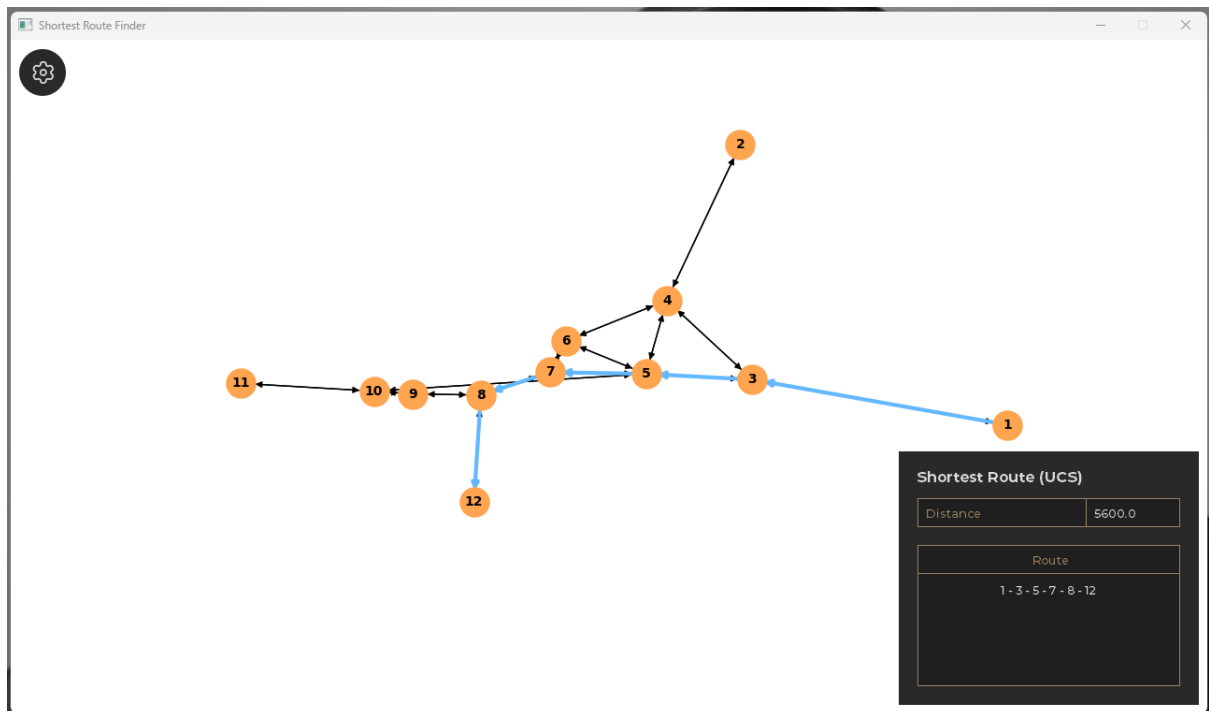
Input: graph1.txt

Starting node = 1

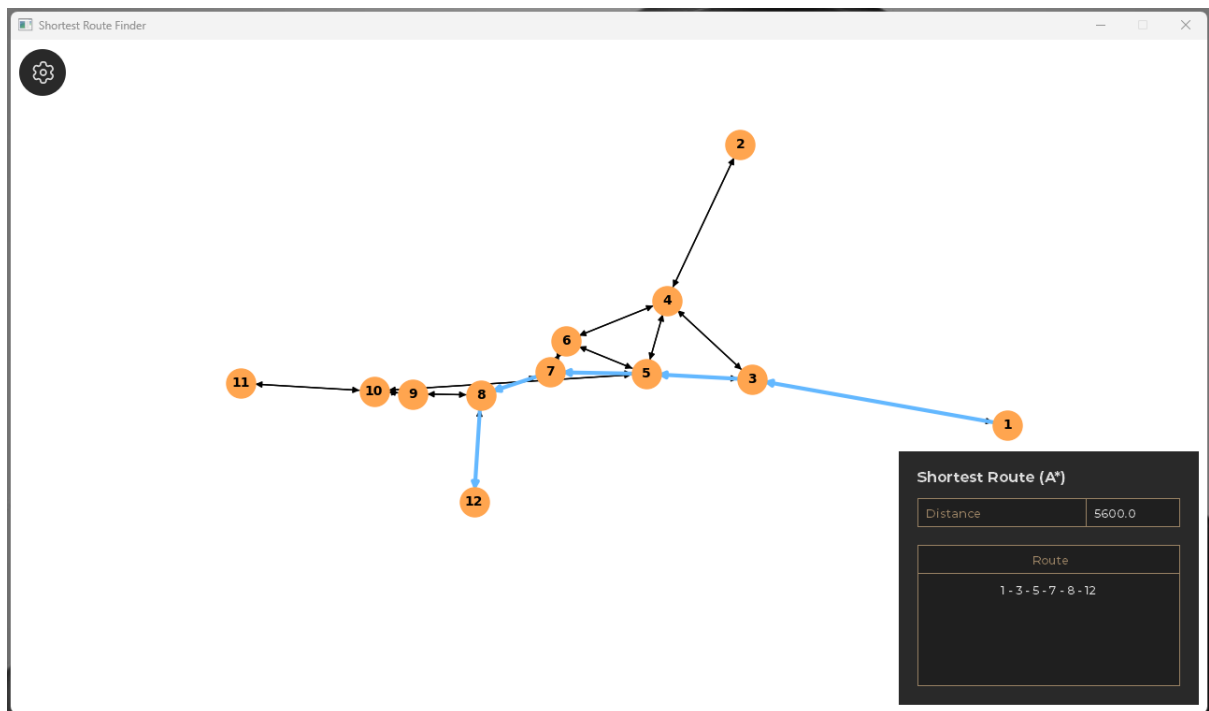
Goal node = 12

```
0 0 2000 0 0 0 0 0 0 0 0 0
0 0 0 2500 0 0 0 0 0 0 0 0
2000 0 0 1500 800 0 0 0 0 0 0 0
0 2500 1500 0 1200 1000 0 0 0 0 0 0
0 0 800 1200 0 900 700 0 0 2000 0 0
0 0 0 1000 900 0 500 0 0 0 0 0
0 0 0 0 700 500 0 600 0 0 0 0
0 0 0 0 0 600 0 500 0 0 1500 0
0 0 0 0 0 0 500 0 300 0 0 0
0 0 0 0 2000 0 0 0 300 0 1000 0
0 0 0 0 0 0 0 0 1000 0 0 0
0 0 0 0 0 0 0 1500 0 0 0 0
```

Output UCS:



Output A*:

**ITB Ganesha**

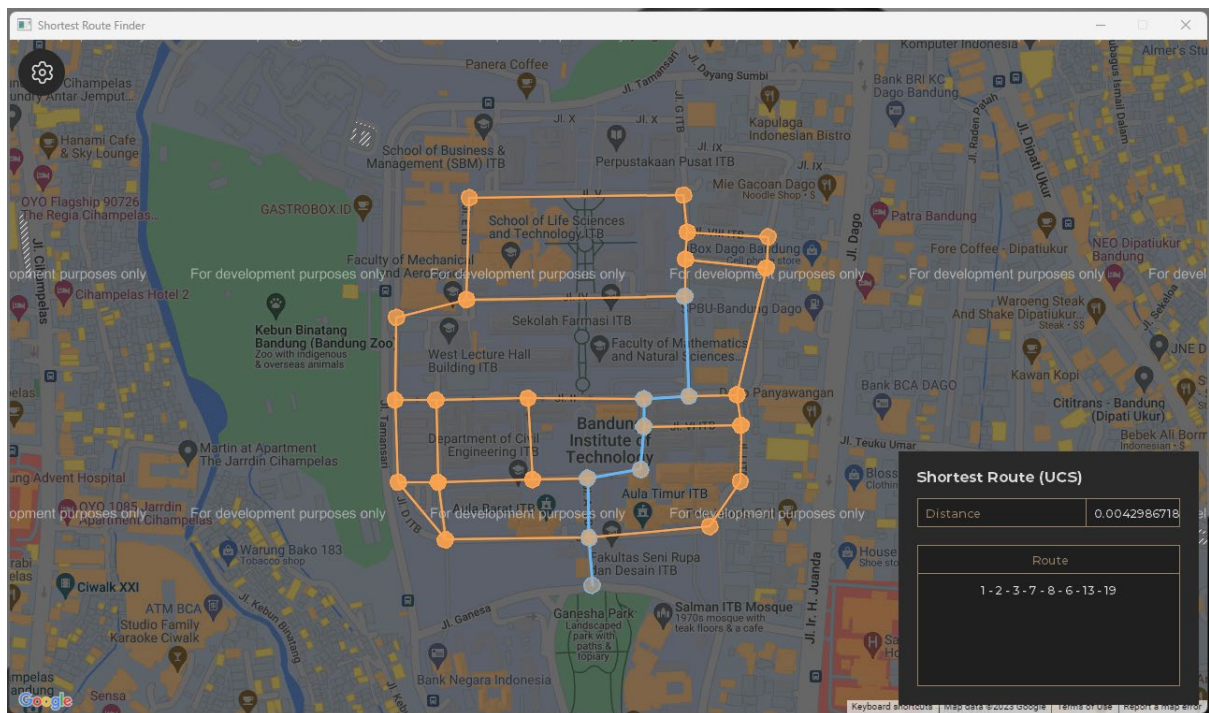
Input: map_itb.txt

Starting node: 1

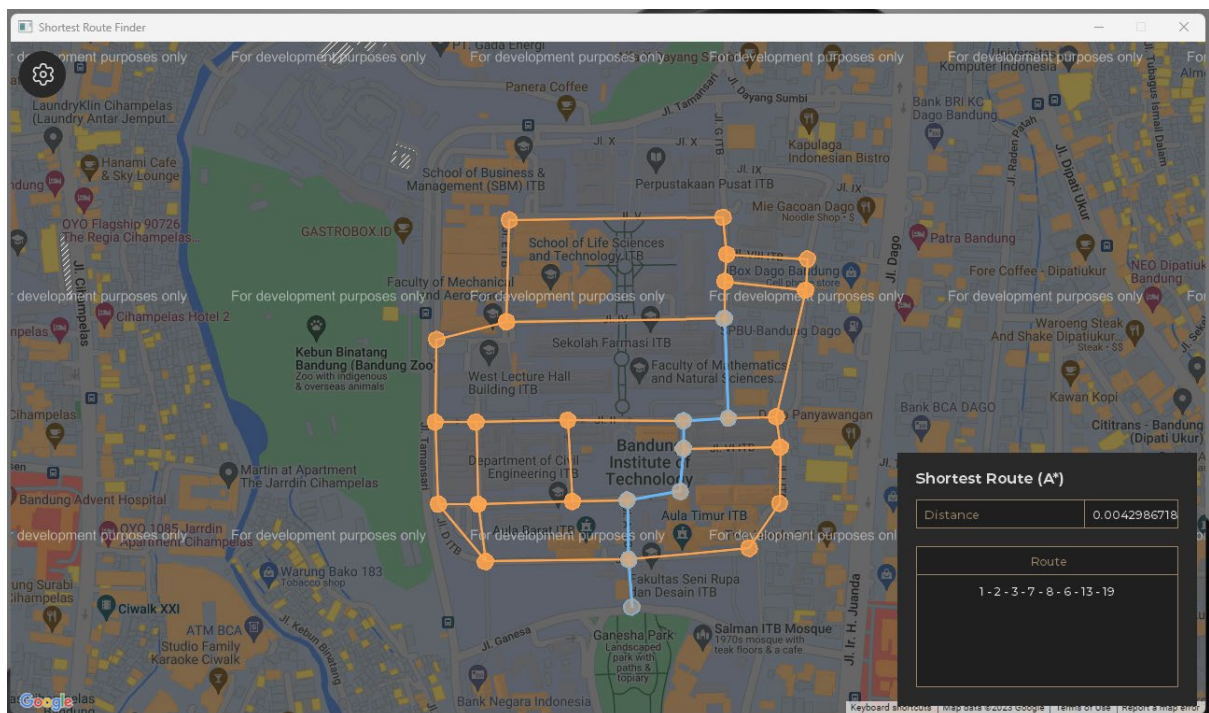
Goal node: 19

[illegible]

Output UCS:



Output A*:



Alun-Alun Bandung

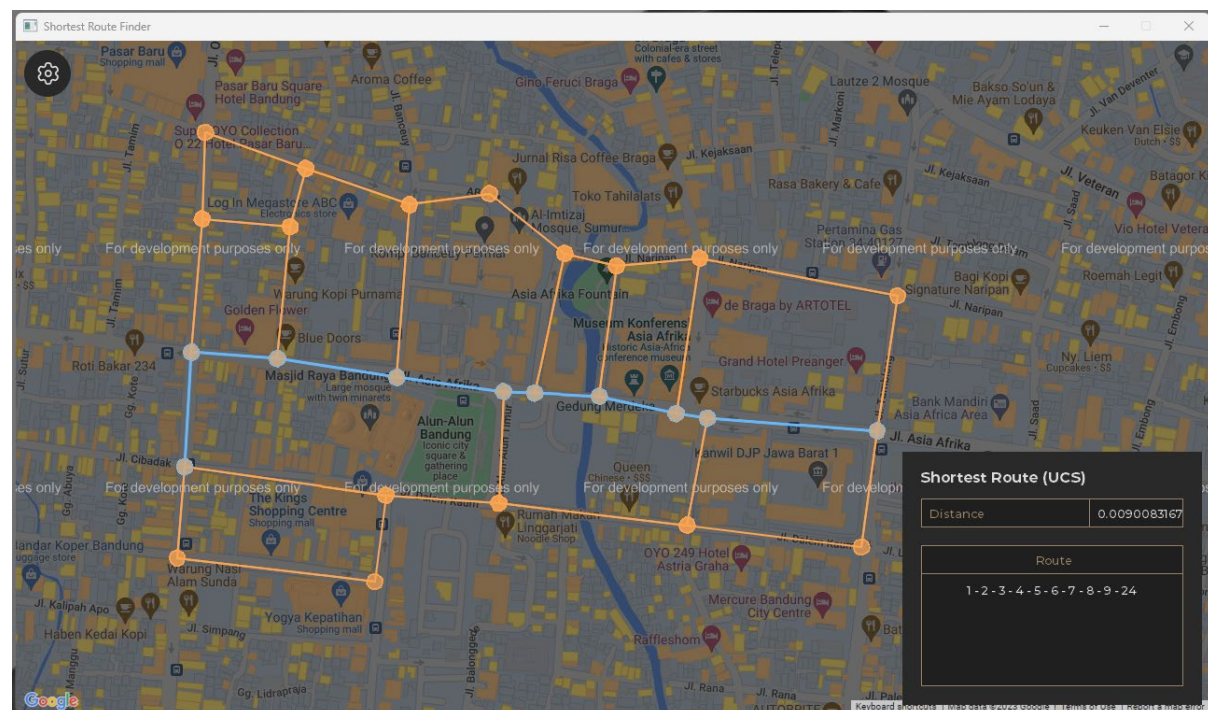
Input: map_alun-alun.txt

Starting node: 1

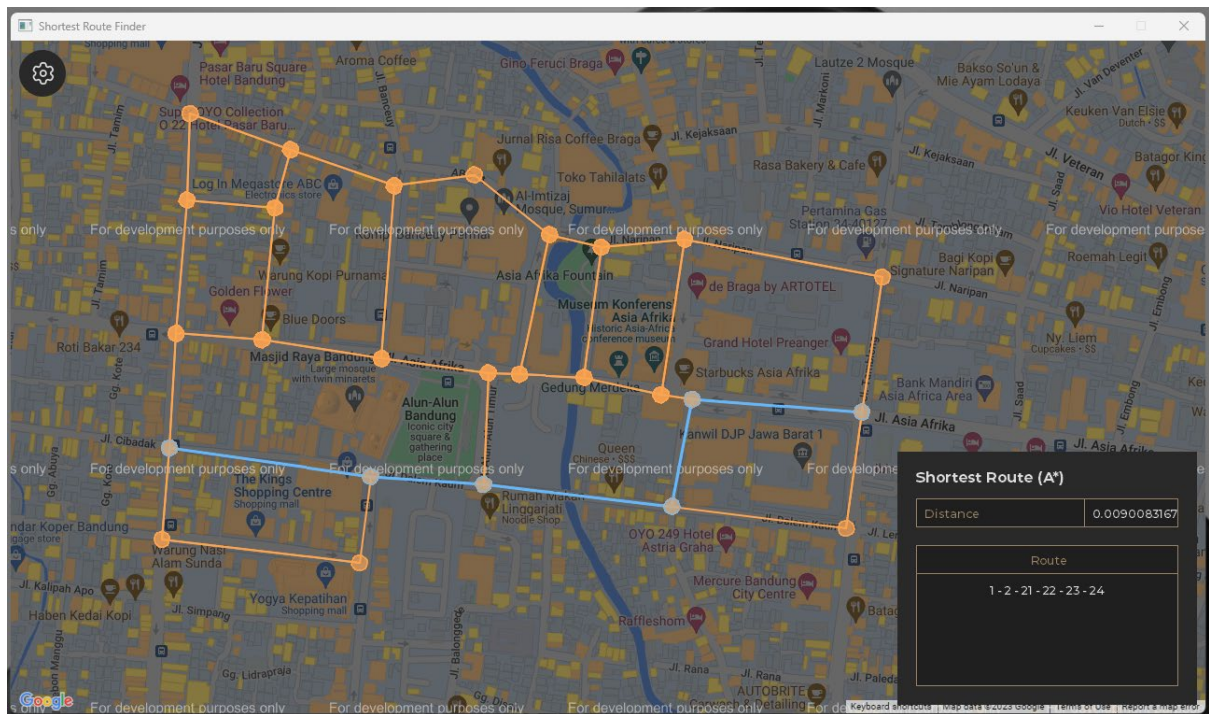
Goal node: 24

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	-6.921693792738858	107.61197418147769
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-6.9215504953333005	107.61002546409011
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-6.921496758794995	107.60966459049982
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-6.921299724768896	107.60878045020361
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-6.921263900391681	107.60804065934352
0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-6.921245988202043	107.60767978575322
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.921084778464792	107.60645281554623
0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.9208698320628015	107.60508149590314
0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-6.920798183240383	107.60408909352984
0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.919275643193241	107.60421539928645
0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.919365204508405	107.60522584533923
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.918290467605133	107.60425148664547
0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.918702450373868	107.60540628213441
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.9191144327831235	107.60659716498236
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.918989046870536	107.60751739263759
0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-6.91966971285289	107.6083834892543
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	-6.919813010829407	107.60897893067829
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	-6.919723449599178	107.60993524569255
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-6.920153343349283	107.61220874931139
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-6.923019291677569	107.61179374468254
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	-6.9227685218936355	107.60979089625644
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	-6.922517751976437	107.60762565471468
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	-6.922428191259422	107.60632650978964
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	-6.9221057725374635	107.60401691881178
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	-6.923144676519569	107.60392670041419
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	-6.923413358211649	107.60620020403304

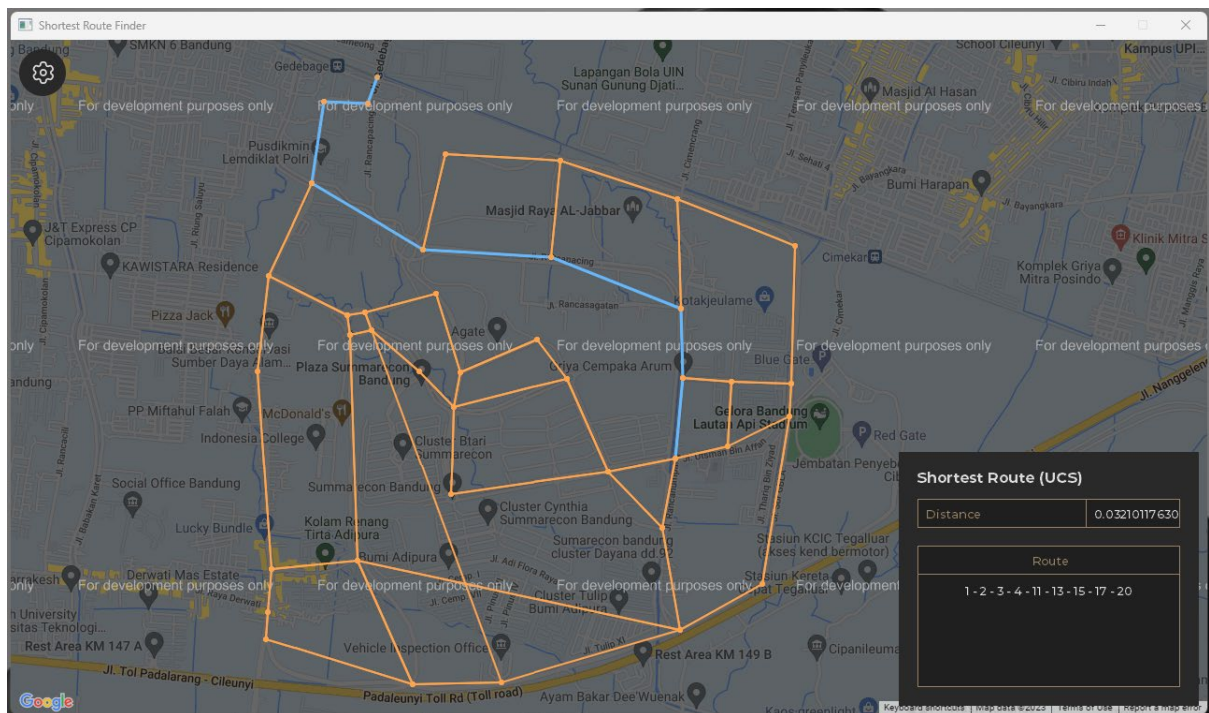
Output UCS:



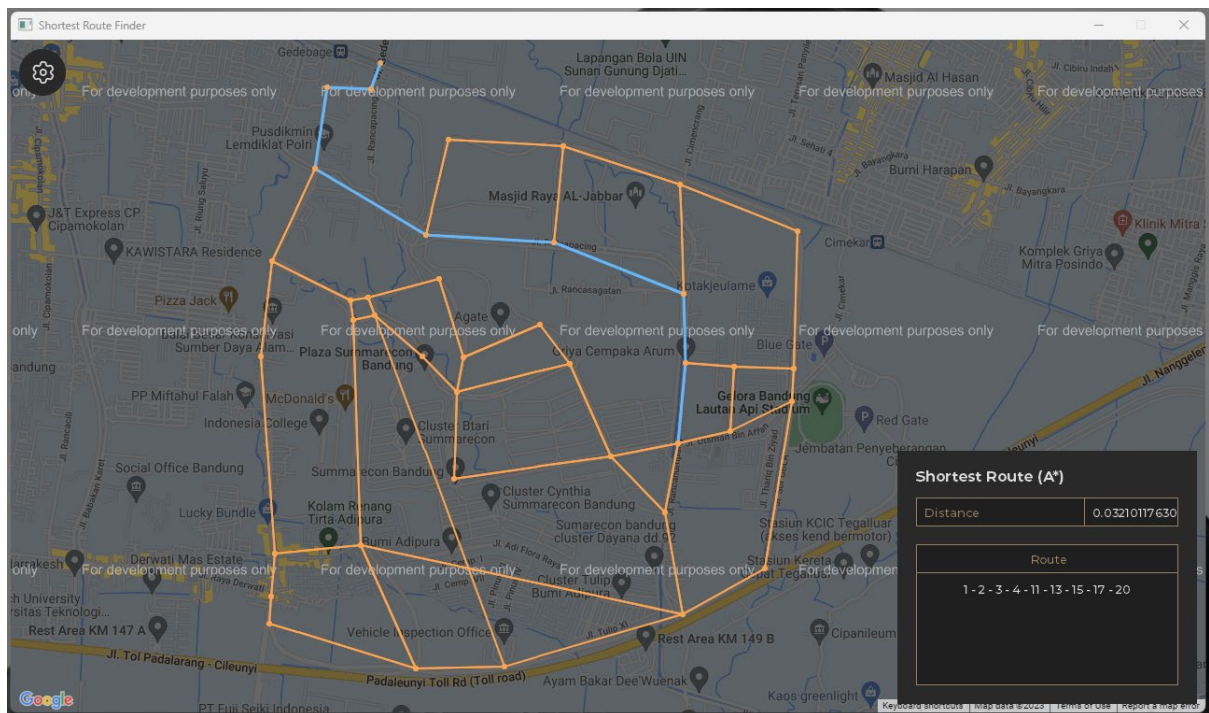
Output A*:



Output UCS:



Output A*:



Puri Dago

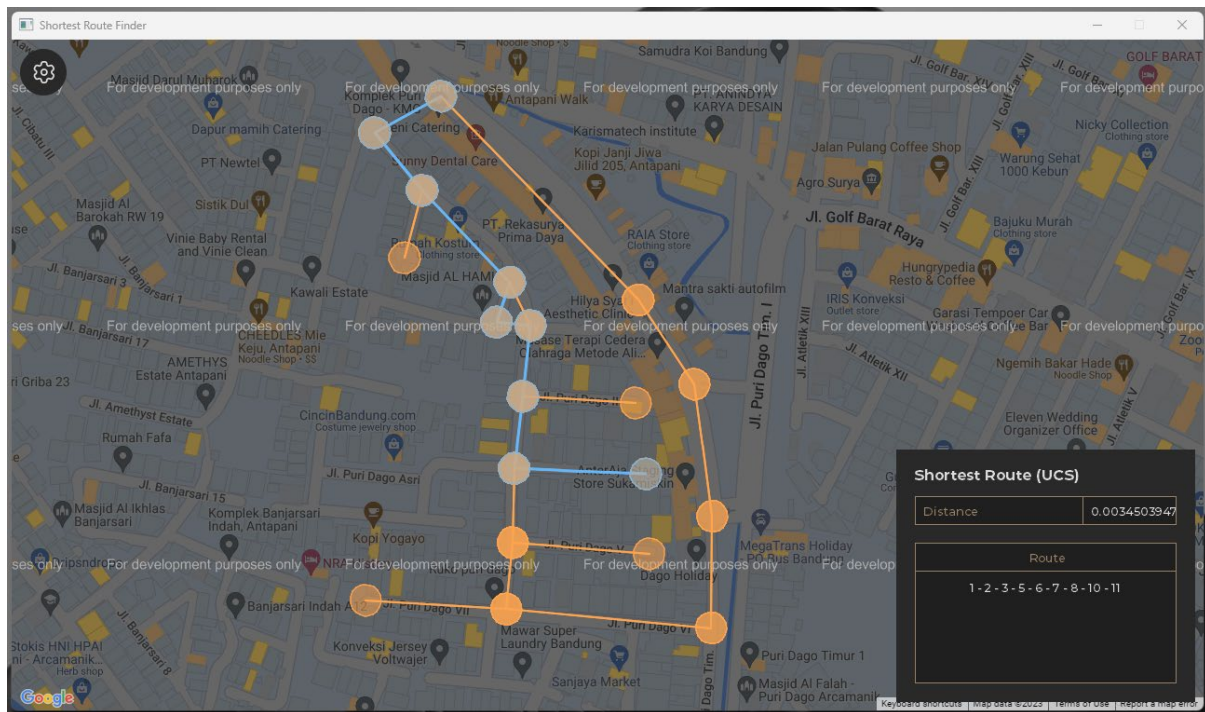
Input: map_puri_dago.txt

Starting node: 1

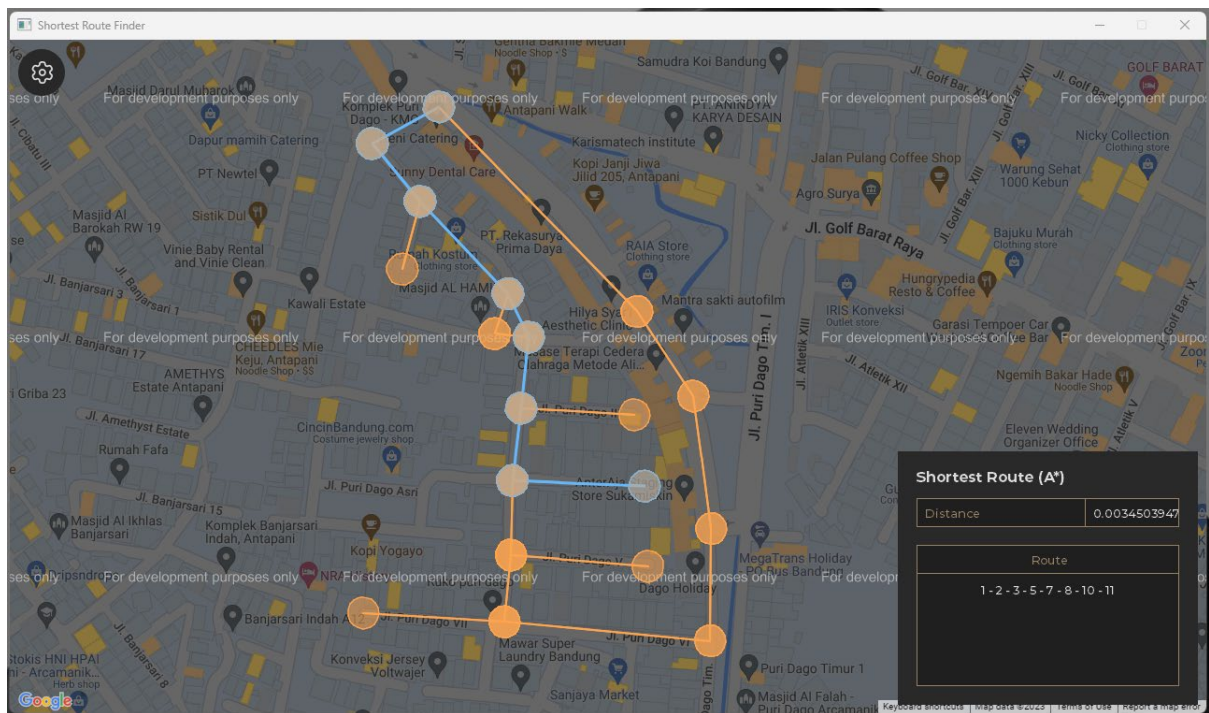
Goal node: 11

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-6.91625368796845	107.66807053929479
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.916462676343243	107.66768777505051
0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.916791991170412	107.66796208942557
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.917178302501934	107.66786001896044
0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.917317627822643	107.66846606234719
0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.917545614622436	107.66838950949835
0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-6.917564613517447	107.66858727102455
0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	-6.917969923095667	107.66854261519605
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-6.9180079208507905	107.6691933144113
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	-6.918381565279984	107.66849157996349
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	-6.918413230048527	107.66925072904795
0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	-6.918805873002069	107.66848520055942
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-6.918869202480124	107.66926986726016
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	-6.919185849742991	107.66844692413497
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-6.919135186195209	107.66763673981794
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	-6.9192935097639925	107.66962711388815
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	-6.918653882220063	107.66963349329221
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	-6.917900260536669	107.66953142282708
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	-6.9174189553028125	107.66920607321946

Output UCS:



Output A*:



Griya Cilegon

Input: map_griya_cilegon.txt

Starting node: 1

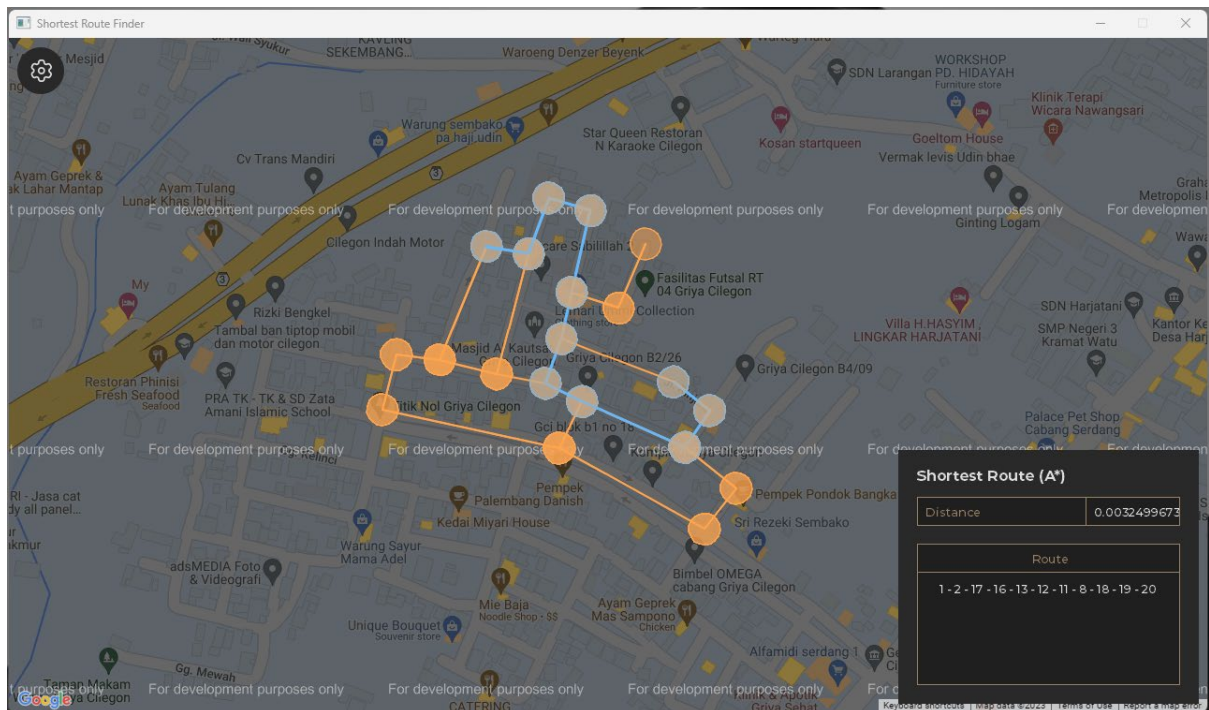
Goal node: 20

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.047227	106.070944
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	-6.047269341407401	106.07118670031738
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.047872603887097	106.07067626008707
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.047953693339958	106.07100258872181
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.047843696310224	106.07042758305991
0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-6.048157941747198	106.07034779499648
0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	-6.048376103587748	106.0713611468614
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	-6.0481110725346525	106.07149193643612
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	-6.048837453644091	106.07219277115138
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	-6.04860573781875	106.07237522512605
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	-6.048006681503873	106.07128230543422
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-6.047753600919982	106.07137354023034
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	-6.047489136337454	106.0714325651138
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	-6.047578354866287	106.07170171859106
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-6.047213752135959	106.07185730494304
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	-6.047023482614647	106.07154000909037
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-6.046951316016145	106.07130086097877
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	-6.048372478302556	106.07208024624735
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	-6.0481625060314546	106.07222369768758
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	-6.047999016112289	106.0720141618771






Output UCS:



Output A*:



LAMPIRAN

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	

REFERENSI

How to create a custom Google Map with Route Planner and Location Markers - [Google Maps Tutorial]

<https://youtu.be/hj9qnKz9NPc>

Plotting Google Map using gmplot Package in Python

<https://www.javatpoint.com/plotting-google-map-using-gmplot-package-in-python>

Network visualization with NetworkX Tutorial

<https://soniakopel.wordpress.com/2017/11/15/network-visualization-with-networkx-tutorial/>

A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial

<https://youtu.be/JtiK0DOeI4A>

Slide Kuliah Penentuan Rute Bagian 1: BFS, DFS, UCS, Greedy Best First Search Bahan Kuliah IF2211 Strategi Algoritma oleh Nur Ulfa Maulidevi

Slide Kuliah Penentuan Rute Bagian 2: Algoritma A* Bahan Kuliah IF2211 Strategi Algoritma oleh Nur Ulfa Maulidevi

Levitin, Anany, Introduction to The Design and Analysis of Algorithms, 3rd ed, USA: Addison-Wesley, 2012