LAPORAN TUGAS KECIL 3
IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Disusun oleh:
1. 13521048 - M Farrel Danendra Rachim
2. 13521074 - Eugene Yap Jin Quan

Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
IF2211 - Strategi Algoritma

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

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, kami diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Jalan dapat diasumsikan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

Langkah pertama dari pencarian di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, program kemudian menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

Berdasarkan deskripsi di atas, batasan dari implementasi adalah sebagai berikut:

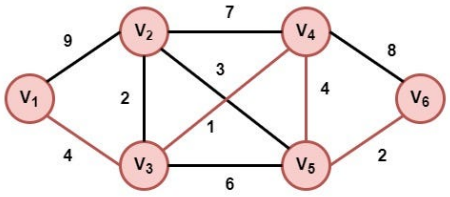
- Implementasi menggunakan bahasa C/C++/Java/Python/C#/Golang/Javascript/Ruby/Typescript/Kotlin
- Hasil implementasi berupa desktop/web based/mobile application
- Solusi menggunakan algoritma *UCS* dan *A*-search*
- Masukan program adalah *file* graf yang merepresentasikan lokasi pada peta
- Program juga menerima masukan simpul asal dan simpul tujuan pada peta pencarian
- Luaran program adalah solusi rute pencarian serta penampilan graf/peta
- Implementasi boleh menggunakan antarmuka GUI atau CLI
- Program juga boleh menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta.

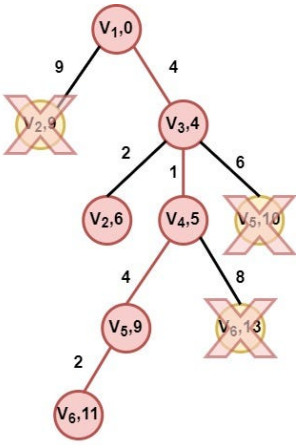
Untuk Tugas Kecil ini, implementasi kami menggunakan bahasa Python.

BAB 2
TEORI DASAR

2.1. Algoritma *Uniform Cost Search (UCS)*

Algoritma UCS merupakan algoritma pencarian graf berdasarkan biaya/berat tiap sisi (*edge*) dari akar graf tersebut. Prinsip kerja algoritma ini mirip dengan BFS, namun algoritma UCS memprioritaskan jalur dengan biaya kumulatif paling kecil setiap kali algoritma melakukan ekspansi nodenya. Total biaya jalur dari akar ke simpul tujuan dilambangkan dengan persamaan f(n) = g(n). Berikut salah satu contoh implementasi algoritma UCS.





Pace	Opened	Closed
0	{(V_1,0)}	{-}
1	{(V_2,9),(V_3,4)}	{(V_1,0)}
2	{(V_2,6),(V_4,5),(V_5,10)}	{(V_1,0),(V_3,4)}
3	{(V_2,6),(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5)}
4	{(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6)}
5	{(V_6,11)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}
6	{-}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}

Path: V_1 - V_3 - V_4 - V_5 - V_6

Total Cost: 11

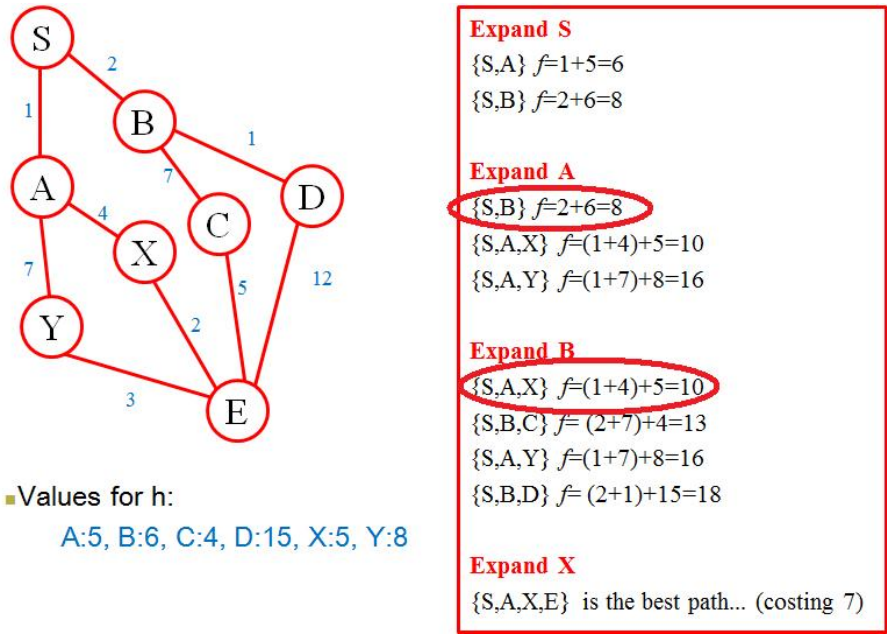
Gambar 2.1.1 Permasalahan Algoritma UCS. Sumber: https://plainenglish.io/blog/uniform-cost-search-ucs-algorithm-in-python-ec3ee03fca9f

Pada contoh di atas, pencarian dimulai dari simpul V1. Ekspansi lalu dilakukan ke simpul V2 dan V3. Karena jalur dari V1 ke V3 memiliki nilai sisi yang lebih kecil, V3 dipilih untuk ekspansi selanjutnya. Dari ekspansi tersebut, ditemukan bahwa V3 terhubung ke V2 dengan nilai sisi yang lebih kecil dari V2 yang terhubung ke V1. Jadi, node (V2, 9) diganti dengan node (V2, 6). Proses ini dilakukan terus menerus sampai diraih node tujuan, yaitu V6. Hasil akhir adalah jalur dengan total biaya terkecil.

2.2. Algoritma Pencarian *A* (A-star Search)*

Algoritma A* juga merupakan salah satu algoritma traversal graf. Algoritma ini dikembangkan dari algoritma Dijkstra dan BFS dengan tujuan mencari jalur dengan biaya kumulatif paling kecil dari simpul awal sampai simpul tujuan. Algoritma A* memiliki persamaan f(n) = g(n) + h(n), di mana f(n) adalah fungsi evaluasi heuristik,

g(n) adalah biaya sejauh ini untuk meraih n, dan h(n) adalah estimasi biaya dari n ke simpul tujuan. Berikut salah satu contoh implementasi algoritma A*.



Gambar 2.2.1 Permasalahan Algoritma A. Sumber: https://stackoverflow.com/questions/5849667/a-search-algorithm*

Pencarian dimulai dari simpul S. Dari simpul S, dilakukan ekspansi ke simpul A dan B. Karena nilai f(A) lebih kecil dari f(B), ekspansi pada simpul A dilakukan terlebih dahulu. Dari ekspansi tersebut, diperoleh simpul {S, B} yang memiliki nilai f = g + h paling kecil. Ekspansi kemudian dilakukan dari situ sampai akhirnya ketemu simpul X.

2.3. Pencarian Rute pada Peta dengan UCS dan A-star Search

Pencarian rute pada peta dengan algoritma UCS dan A-star search dapat dilakukan pemetaan elemen-elemen pada peta ke elemen algoritma pencarian. Berikut adalah pemetaan elemen-elemen tersebut ke elemen algoritma pencarian:

- Peta: Graf yang digunakan dalam pencarian
- Titik-titik lokasi relevan: Simpul dalam graf pencarian
- Jalan penghubung titik-titik lokasi: Sisi dalam graf pencarian
- Jarak antara titik: Bobot sisi

Adapun elemen urutan pencarian sebuah simpul dalam algoritma UCS dan A* Search. Urutan prioritas dapat dihitung berdasarkan rumus f(n)=g(n)+h(n). Keterangan:

- f(n): prioritas pencarian lokasi n
- g(n): total bobot dari rute yang telah dilalui menuju n
- h(n): nilai heuristik, untuk UCS bernilai 0, untuk A* Search dapat menggunakan jarak lurus antara kedua titik lokasi

BAB 3
IMPLEMENTASI PROGRAM

3.1. File: route_planner.py

```
"""
:file: route_planner.py

Used to find the shortest distance between two points
"""

from queue import PriorityQueue
from graph import LocationGraph
```

```
class RoutePlanner:
    graph = LocationGraph()
    solution_cost = float('inf')
    solution_path = []
    with_astar_heuristic = False
    show_debug = False

    def __init__(self, graph, with_astar_heuristic,
show_debug=False):
        """
        :param graph: LocationGraph
        :param with_astar_heuristic: astar heuristic boolean
toggle, determines whether UCS or Astar is used
        :param show_debug: debug message toggle
        """
        self.graph = graph
        self.with_astar_heuristic = with_astar_heuristic
        self.show_debug = show_debug
```

```
    class SearchNode:
        def __init__(self, name, path_list, path_cost):
            """
            :param name: string of instance location name
            :param path_list: list of location name strings
'visited' by instance
            :param path_cost: total cost of path taken by instance
            """
            self.name = name
            self.path_list = [location for location in path_list]
            self.path_cost = path_cost
```

```
        def add_self_to_path_list(self):
            """
            marks instances' name as 'visited' in instances' path
history
            """
            self.path_list = self.path_list + [self.name]

        def __lt__(self, other):
            """
            less than operator overload, used for enqueue

            :param other: another SearchNode instance
            :return: boolean value of whether self is less than
other
            """
            # less than comparator
            return self.path_cost < other.path_cost
```

```
    def plan_route(self, start_node_name, finish_node_name):
        """UCS/AStar Search Algorithm
        :param start_node_name: string of starting node name
        :param finish_node_name: string of finish node (goal)

        :return: boolean of search success
        """

        # SETUP: priority queue, starting node, explored nodes
        search_pqueue = PriorityQueue()
        starting_node = self.SearchNode(start_node_name, [], 0)

        # enqueue start_node; if search is using astar, priority
considers the heuristic value
        if self.with_astar_heuristic is True:
            node_priority = starting_node.path_cost +
self.graph.get_distance_between(starting_node.name,

finish_node_name)
        else:
            node_priority = starting_node.path_cost

        search_pqueue.put((node_priority, starting_node))
        explored_node_names = set()
```

```
        # SEARCH: do search loop
        while not search_pqueue.empty():
            # dequeue current node to search
            node_priority, current_node = search_pqueue.get()
            current_node.add_self_to_path_list()

            # show debug messages
            if self.show_debug is True:
                print(node_priority, "---", current_node.name,
"--", current_node.path_cost, current_node.path_list)

            # Goal check: return from function if goal is met
            if current_node.name == finish_node_name:
                self.solution_path = current_node.path_list
                self.solution_cost = current_node.path_cost

                return True  # search success

            # mark current node name as visited
            explored_node_names.add(current_node.name)

            # enqueue neighbors
            for neighbor_name in
self.graph.get_neighbors(current_node.name):
                if neighbor_name not in explored_node_names:
                    neighbor_cost = current_node.path_cost +
self.graph.get_edge_cost(current_node.name, neighbor_name)

                    new_neighbor_node =
self.SearchNode(neighbor_name, current_node.path_list,
neighbor_cost)

                    # enqueue neighbor to search queue; if search
is using astar, priority considers the heuristic value
                    if self.with_astar_heuristic is True:
                        node_priority = neighbor_cost +
self.graph.get_distance_between(neighbor_name, finish_node_name)
                    else:
                        node_priority = neighbor_cost

                    search_pqueue.put((node_priority,
new_neighbor_node))

        # Search failed
```

```
        return False
```

```
    def get_solution(self):
        """
        :return: tuple of instance's latest solution cost and
solution path
        """
        return self.solution_cost, self.solution_path
```

```
    def print_solution(self):
        """
        prints solution route taken and cost of route
        """
        print(f"Rute = {self.solution_path}")
        print(f"Biaya total = {self.solution_cost}")
```

```
    def reset_solution(self):
        """
        clears saved solution
        """
        self.solution_cost = float('inf')
        self.solution_path = []
```

3.2. File: utils.py

```
"""
:file: utils.py

Some methods to support implementations of other classes
"""

from math import *
```

```
def haversine_distance(lat1, lon1, lat2, lon2):
    """
    calculates haversine distance between two geographical points

    :param lat1: latitude of first point
    :param lon1: longitude of first point
    :param lat2: latitude of second point
    :param lon2: longitude of second point
    :return: distance between two point
    """
    lat1 = radians(lat1)
    lon1 = radians(lon1)
```

```
    lat2 = radians(lat2)
    lon2 = radians(lon2)
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2)
** 2
    return 6371 * (2 * asin(sqrt(a)))
```

```
def is_square(matrix, n):
    """
    check if a matrix is square

    :param matrix:
    :param n: size of matrix to be validated
    :return: boolean of validity
    """
    return all(len(i) == n for i in matrix) and len(matrix) == n
```

```
def validate_int_input(min_value, max_value, choices=""):
    """
    loop and validate integer input between a range

    :param min_value: minimum value of range
    :param max_value: maximum value of range
    :param choices: choices prompt
    :return: valid integer
    """
    print(choices)

    is_valid = False
    input_value = 0
    while not is_valid:
        input_value = input("Ketik angka: ")
        try:
            input_value = int(input_value)
            if min_value <= input_value <= max_value:
                is_valid = True
            else:
                print("Masukan tidak valid")
        except ValueError:
            print("Masukan tidak valid")

    return input_value
```

3.3. File: graph_reader.py

```
"""
:file: graph_reader.py

Read and gather data about coordinates and matrix from a text file
"""

import os
from utils import *
from graph import LocationGraph
```

```
class GraphReader:
    def __init__(self, use_weighted_matrix=False):
        """

        :param use_weighted_matrix: boolean toggle for using
haversine distance or matrix for edge weights
        """
        self.latitude = []
        self.longitude = []
        self.location_name = []
        self.coordinate_tuple = []
        self.adj_matrix = []
        self.filename = ""
        self.use_weighted_matrix = use_weighted_matrix
```

```
    class GraphReaderException():
        pass
```

```
    def read_graph_file(self):
        """
        iterates _read_file until success
        """
        successful_read = False
        while not successful_read:
            try:
                self._read_file()
                successful_read = True
            except self.GraphReaderException as reader_exception:
                print(reader_exception.args[0])
```

```
    def _read_file(self):
        """
        helper method for read_graph_file, reads graph txt file
```

```
        """
        # Detect file
        path = os.path.realpath(__file__)
        directory = os.path.dirname(path)
        directory = directory.replace('src', 'test')
        os.chdir(directory)
        directory += "/"

        print("Masukkan nama file (dari folder test): ", end="")
        filename = input()
        filename = directory + filename
        if not (os.path.isfile(filename)):
            raise self.GraphReaderException("File tidak
ditemukan.")

        # Read all of file
        with open(filename, 'r', encoding='utf-8') as file:
            lines = file.read()
            result = lines.split('\n~\n')  # Pembacaan koordinat
dan adjMatrix dipisahkan dengan simbol '~'

        # Read coordinates
        result_coordinates = result[0]
        node_coordinates = result_coordinates.splitlines(False)

        # Put names, latitudes, longitudes in separate lists
        location_name = []
        latitude = []
        longitude = []
        for coord in node_coordinates:
            splited_line = coord.split()
            if len(splited_line) == 3:
                if splited_line[0] in location_name:
                    raise self.GraphReaderException("Nama simpul
harus unik.")

                location_name.append(splited_line[0])
                try:
                    latitude.append(float(splited_line[1]))
                    longitude.append(float(splited_line[2]))
                except ValueError:
                    raise self.GraphReaderException("Koordinat
harus berupa float.")
            else:
```

```
                raise self.GraphReaderException(
                    "Simpul harus didefinisikan dalam format
'<nama> <latitude> <longitude>' sebelum mendefinisikan "
                    "matriks.")

        # Add tuple of names, latitudes, longitudes
        coordinate_tuple = []
        for i in range(len(latitude)):
            coordinate_tuple.append((location_name[i], latitude[i],
longitude[i]))

        # Read adjacency matrix
        try:
            result_matrix = result[1]
        except IndexError:
            raise self.GraphReaderException("File tidak berisi
pemisah '~', matriks ketetanggaan tidak dapat dibaca.")

        matrix_rows = result_matrix.splitlines(False)
        try:
            adj_matrix = [[float(num) for num in row.split(' ')]
for row in matrix_rows]
        except ValueError:
            raise self.GraphReaderException("Matriks harus berisi
angka.")

        if not is_square(adj_matrix, len(location_name)):
            raise self.GraphReaderException(
                "Matriks ketetanggaan harus persegi dan berukuran
sama dengan jumlah simpul.")
        else:
            # Calculate distance for nodes that are not valued 0
            for i in range(len(adj_matrix)):
                for j in range(len(adj_matrix[i])):
                    if adj_matrix[i][j] > 0:
                        if self.use_weighted_matrix is False:
                            # calculate haversine distance if
use_weighted_matrix disabled, else use matrix as defined
                            adj_matrix[i][j] =
haversine_distance(latitude[i], longitude[i], latitude[j],
longitude[j])
                    elif adj_matrix[i][j] < 0:
                        raise self.GraphReaderException("Elemen
matriks harus bernilai 0 atau positif.")
```

```
        # commit read
        self.latitude = latitude
        self.longitude = longitude
        self.location_name = location_name
        self.coordinate_tuple = coordinate_tuple
        self.adj_matrix = adj_matrix
        self.filename = filename
```

```
    def print_reader_info(self):
        """
        prints location names
        """
        print("Nama lokasi:")
        for i in range(len(self.location_name)):
            print("%d. %s" % (i + 1, self.location_name[i]))
```

```
    def get_location_graph(self):
        """
        converts read file to LocationGraph
        :return: LocationGraph object
        """
        result = LocationGraph()

        # add nodes
        location_count = len(self.location_name)
        for i in range(location_count):
            result.add_node(self.location_name[i],
self.latitude[i], self.longitude[i])

        # add edges
        for i in range(location_count):
            for j in range(location_count):
                edge_weight = self.adj_matrix[i][j]
                if edge_weight > 0:
                    result.add_weighted_edge(self.location_name[i],
self.location_name[j], edge_weight)

        return result
```

3.4. File: graph.py

```
"""
:file: graph.py
```

```
a wrapper class for nx.DiGraph (NetworkX's Directed Graph)
"""

import networkx as nx
import matplotlib.pyplot as plt
from utils import *
```

```
class LocationGraph(nx.DiGraph):
    def __init__(self, incoming_graph_data=None, **attr):
        super().__init__(incoming_graph_data, **attr)
```

```
    def add_node(self, node_name, latitude=0, longitude=0):
        """add new node, overrides method from nx.DiGraph

        :param node_name:
        :param latitude:
        :param longitude:
        """
        super().add_node(node_name, lat=latitude, lon=longitude)
```

```
    def add_weighted_edge(self, start_node_name, finish_node_name,
weight):
        """add weighted edge using add_weighted_edges_from method

        :param start_node_name:
        :param finish_node_name:
        :param weight:
        :return:
        """
        return super().add_weighted_edges_from([(start_node_name,
finish_node_name, weight)])
```

```
    def get_neighbors(self, n):
        """get iterator of neighbor of node_name n

        :param n: node name
        :return: iterator to neighbors of n
        """
        # return iterator to neighbors of n
        return super().neighbors(n)
```

```
    def get_edge_cost(self, start_node_name, finish_node_name):
        """get edge cost of an edge, might be 'inf'
```

```
        :param start_node_name:
        :param finish_node_name:
        :return: cost of directed edge
        """
        cost = self.get_edge_data(start_node_name,
finish_node_name)
        if cost is None:
            return float('inf')
        else:
            return cost['weight']
```

```
    def get_distance_between(self, start_node_name,
finish_node_name):
        """haversine distance between two nodes, used in astar
heuristic

        :param start_node_name:
        :param finish_node_name:
        :return: haversine distance
        """
        # check for invalid node parameter
        if (not self.has_node(start_node_name)) or (not
self.has_node(finish_node_name)):
            return float('inf')
        else:
            # haversine distance
            node_1 = self.nodes[start_node_name]
            node_2 = self.nodes[finish_node_name]

            return haversine_distance(node_1['lat'], node_1['lon'],
node_2['lat'], node_2['lon'])
```

```
    def display_graph(self, with_weights=True, solution_path=[]):
        """display graph using nx.draw and matplotlib.pyplot

        :param with_weights:
        :param solution_path:
        """
        node_pos = {n: (d['lon'], d['lat']) for n, d in
self.nodes(data=True)}
        node_index = {n: (idx + 1) for idx, n in
enumerate(self.nodes())}

        if solution_path:
```

```
            # create a list of edge colors
            edge_colors = []
            edge_route_tuples = [(solution_path[i], solution_path[i
+ 1]) for i in range(len(solution_path) - 1)]

            for edge in self.edges():
                if edge in edge_route_tuples:
                    edge_colors.append('y')
                else:
                    edge_colors.append((0.5, 0.5, 0.5, 0.5))  #
transparent gray

            # create a list of node colors
            node_colors = []
            for node in self.nodes():
                if node == solution_path[0]:
                    node_colors.append('g')
                elif node == solution_path[len(solution_path) - 1]:
                    node_colors.append('r')
                elif node in solution_path:
                    node_colors.append('y')
                else:
                    node_colors.append('c')

        else:
            edge_colors = (0.5, 0.5, 0.5, 0.5)  # default:
transparent gray, cyan
            node_colors = 'c'

        # Draw nodes, node indices, edges, solution edges
        nx.draw(self, pos=node_pos, labels=node_index,
edge_color=edge_colors, node_color=node_colors)

        if with_weights is True:
            # Draw edges with weights
            edge_labels = {(u, v): f"{self[u][v]['weight'] : .2f}"
for (u, v) in self.edges()}
            nx.draw_networkx_edge_labels(self, node_pos,
edge_labels=edge_labels, label_pos=0.7)

        plt.axis('off')
        plt.show()
```

3.5. *FIle: main.py*

```
"""
:file: main.py

main program for route planning between two points of a map
"""

from graph import *
from graph_reader import *
from route_planner import *
from utils import *
```

```
# select adjacency matrix mode
weight_choice_prompt = (
    "Pilih mode penentuan bobot sisi:\n"
    "1. Perhitungan Haversine (Koordinat)\n"
    "2. Matriks ketetanggaan"
)
weight_choice = validate_int_input(1, 2, weight_choice_prompt)
use_adj_matrix = False if weight_choice == 1 else True
print()


# select and read map file
reader = GraphReader(use_adj_matrix)
reader.read_graph_file()

# select algorithm choice (UCS/A*)
algorithm_choice_prompt = (
    "Pilih algoritma yang ingin digunakan:\n"
    "1. Algoritma UCS\n"
    "2. Algoritma A*"
)
algorithm_choice = validate_int_input(1, 2,
algorithm_choice_prompt)
use_astar = False if algorithm_choice == 1 else True
print()

# show locations, select start and finish point of search
reader.print_reader_info()
start_index = validate_int_input(1, len(reader.coordinate_tuple),
"Pilih tujuan awal")
finish_index = validate_int_input(1, len(reader.coordinate_tuple),
```

```
"Pilih tujuan akhir")
print()

# start route search
print("Hasil pencarian")
location_graph = reader.get_location_graph()
solver = RoutePlanner(location_graph,
with_astar_heuristic=use_astar, show_debug=False)
found_route = solver.plan_route(reader.location_name[start_index -
1], reader.location_name[finish_index - 1])
if found_route:
    solver.print_solution()
else:
    print("Rute tidak ditemukan")

location_graph.display_graph(with_weights=True,
solution_path=solver.solution_path)
```

BAB 4
EKSPERIMEN

4.1. test1.txt

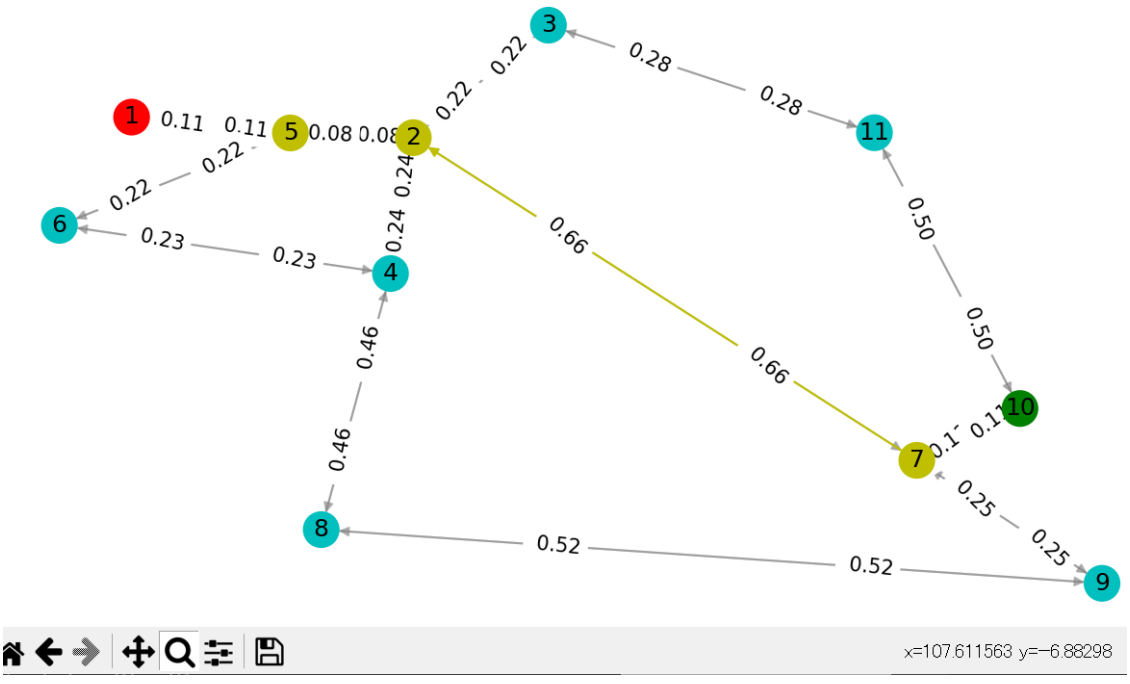
Konfigurasi file (bobot sisi = Haversine)

```
sangkuriang_siliwangi -6.884907 107.612047
simpang_dago -6.885235 107.613702
tubisluar_djuanda -6.883440 107.614496
djuanda_dayangsumbi -6.887404 107.613568
siliwangi_sumurbandung -6.885160 107.612980
sumurbandung_dayangsumbi -6.886635 107.611623
dipatiukur_sekeloa -6.890385 107.616660
djuanda_teukuumar -6.891491 107.613161
teukuumar_dipatiukur -6.892348 107.617748
sekeloa_tubisdalam -6.889558 107.617266
tubisluar_tubisdalam -6.885156 107.616411
~
0 0 0 0 1 0 0 0 0 0 0
0 0 1 1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 1 0 1 0 0 0
1 1 0 0 0 1 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 1 0
0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 1 0 0 0 1
0 0 1 0 0 0 0 0 0 1 0
```

Algoritma UCS (awal = “sekeloa_tubisdalam”, akhir = “sangkuriang_siliwangi”)

Screenshot program dan visualisasi

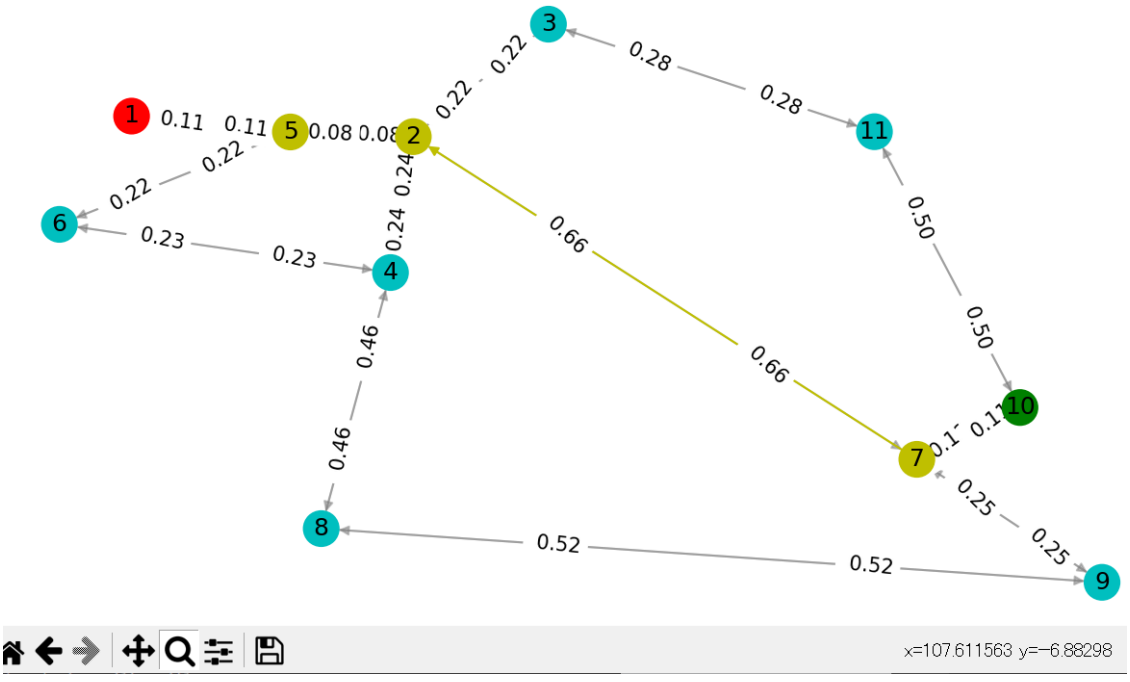
```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1

Masukkan nama file (dari folder test): test1.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 1

Nama lokasi:
1. sangkuriang_siliwangi
2. simpang_dago
3. tubisluar_djuanda
4. djuanda_dayangsumbi
5. siliwangi_sumurbandung
6. sumurbandung_dayangsumbi
7. dipatiukur_sekeloa
8. djuanda_teukuumar
9. teukuumar_dipatiukur
10. sekeloa_tubisdalam
11. tubisluar_tubisdalam
Pilih tujuan awal
Ketik angka: 10
Pilih tujuan akhir
Ketik angka: 1

Hasil pencarian
Rute = ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', 'simpang_dago', 'siliwangi_sumurbandung',
'sangkuriang_siliwangi']
Biaya total = 0.9598383051668624
```



Algoritma A* (awal = “sekeloa_tubisdalam”, akhir = “sangkuriang_siliwangi”)

Screenshot program dan visualisasi

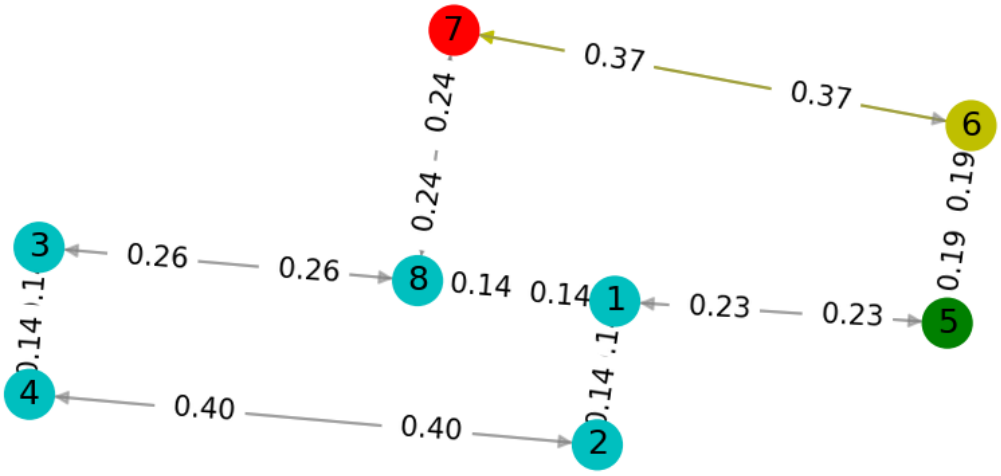
```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1

Masukkan nama file (dari folder test): test1.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 2

Nama lokasi:
1. sangkuriang_siliwangi
2. simpang_dago
3. tubisluar_djuanda
4. djuanda_dayangsumbi
5. siliwangi_sumurbandung
6. sumurbandung_dayangsumbi
7. dipatiukur_sekeloa
8. djuanda_teukuumar
9. teukuumar_dipatiukur
10. sekeloa_tubisdalam
11. tubisluar_tubisdalam
Pilih tujuan awal
Ketik angka: 10
Pilih tujuan akhir
Ketik angka: 1

Hasil pencarian
Rute = ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', 'simpang_dago', 'siliwangi_sumurbandung', 'sangkuriang_siliwangi']
Biaya total = 0.9598383051668624
```



4.2. test2.txt

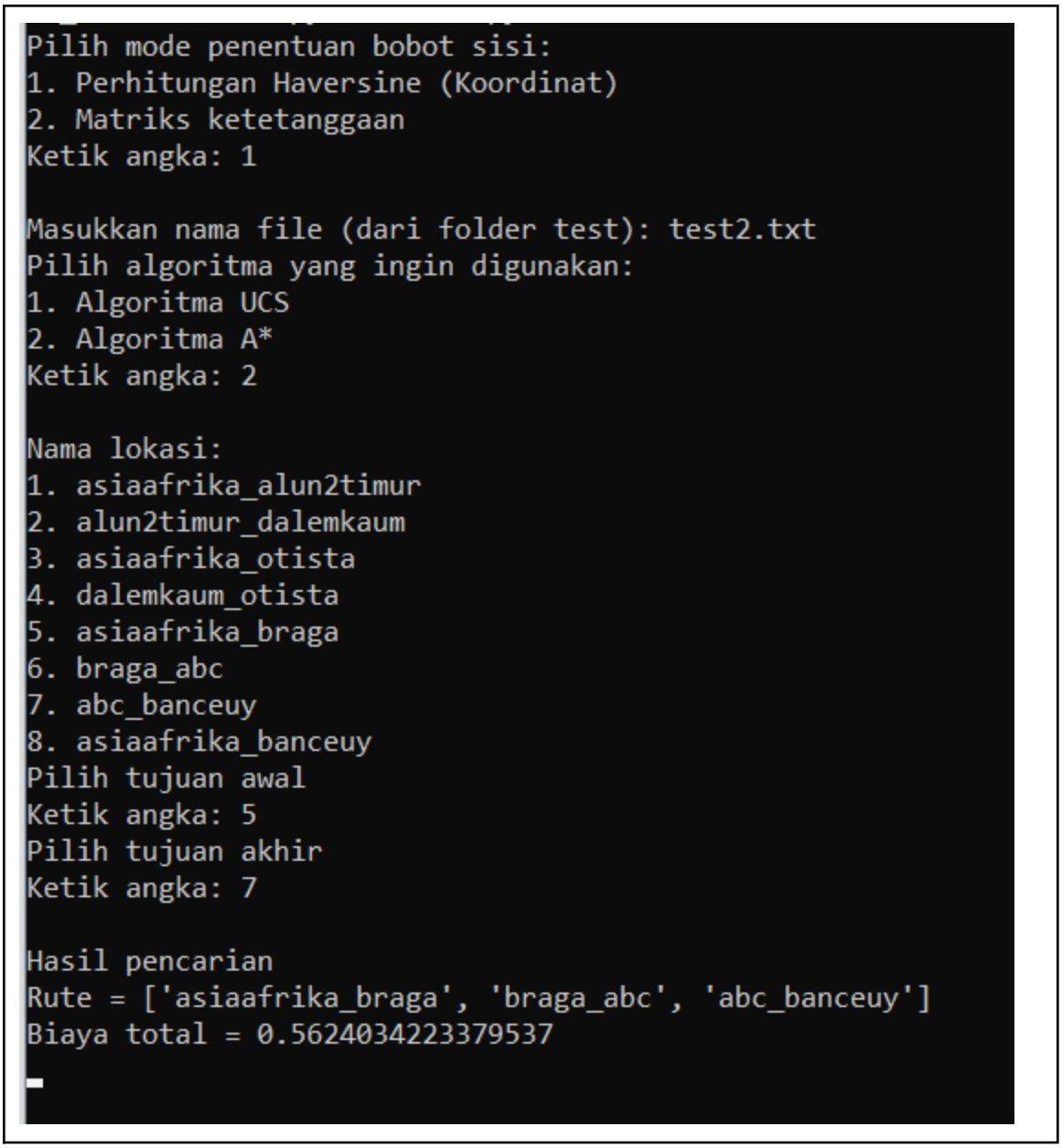
Konfigurasi file (bobot sisi = Haversine)
asiaafrika_alun2timur -6.92121 107.60767 alun2timur_dalemkaum -6.92246 107.60756 asiaafrika_otista -6.92074 107.60405 dalemkaum_otista -6.92202 107.60399 asiaafrika_braga -6.9214 107.60976 braga_abc -6.91968 107.60991 abc_banceuy -6.91885 107.60666 asiaafrika_banceuy -6.92103 107.60643 ~ 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0
Algoritma UCS (awal = “asiaafrika_braga”, akhir = “abc_banceuy”)
Screenshot program dan visualisasi

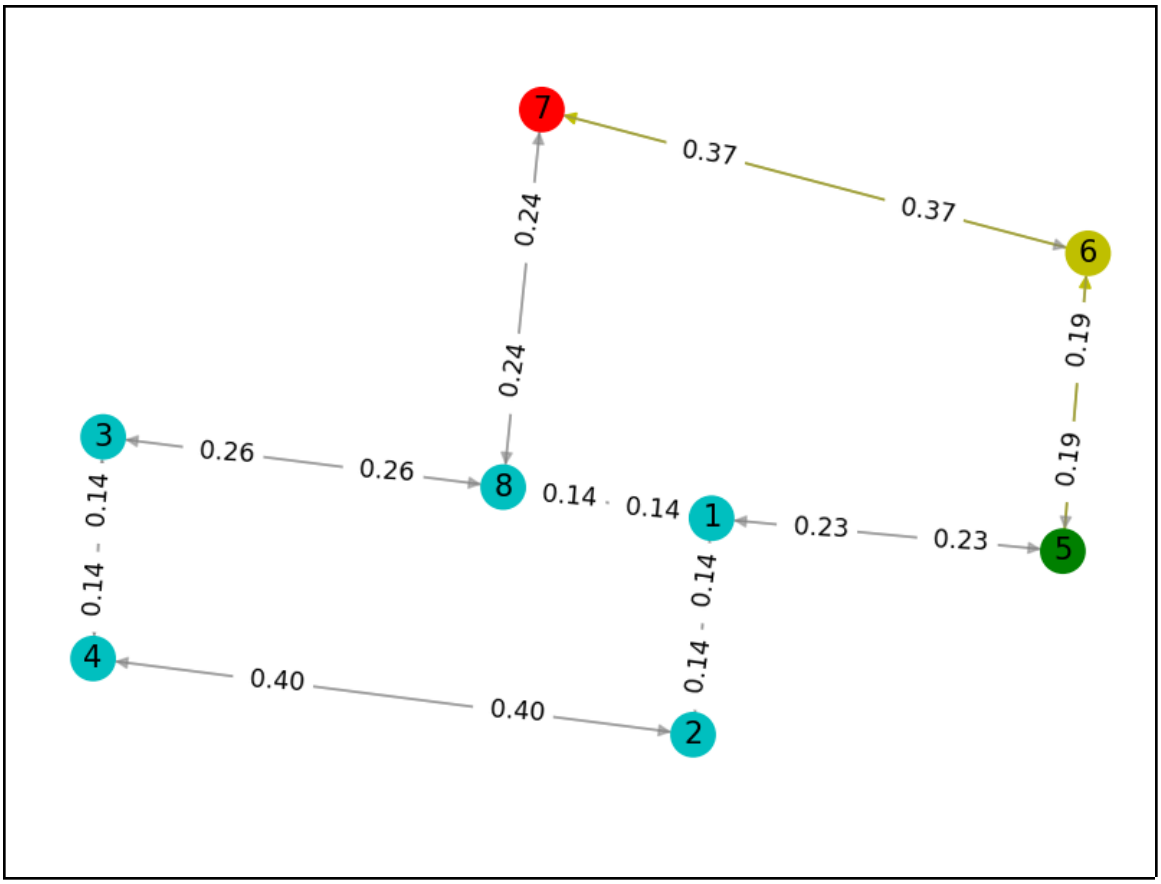
```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1

Masukkan nama file (dari folder test): test2.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 1

Nama lokasi:
1. asiaafrika_alun2timur
2. alun2timur_dalemkaum
3. asiaafrika_otista
4. dalemkaum_otista
5. asiaafrika_braga
6. braga_abc
7. abc_banceuy
8. asiaafrika_banceuy
Pilih tujuan awal
Ketik angka: 5
Pilih tujuan akhir
Ketik angka: 7

Hasil pencarian
Rute = ['asiaafrika_braga', 'braga_abc', 'abc_banceuy']
Biaya total = 0.5624034223379537
```



Algoritma A* (awal = “asiaafrika_braga”, akhir = “abc_banceuy”)

Screenshot program dan visualisasi


Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1
Masukkan nama file (dari folder test): test2.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 2
Nama lokasi:
1. asiaafrika_alun2timur
2. alun2timur_dalemkaum
3. asiaafrika_otista
4. dalemkaum_otista
5. asiaafrika_braga
6. braga_abc
7. abc_banceuy
8. asiaafrika_banceuy
Pilih tujuan awal
Ketik angka: 5
Pilih tujuan akhir
Ketik angka: 7
Hasil pencarian
Rute = ['asiaafrika_braga', 'braga_abc', 'abc_banceuy']
Biaya total = 0.5624034223379537



4.3. test3.txt

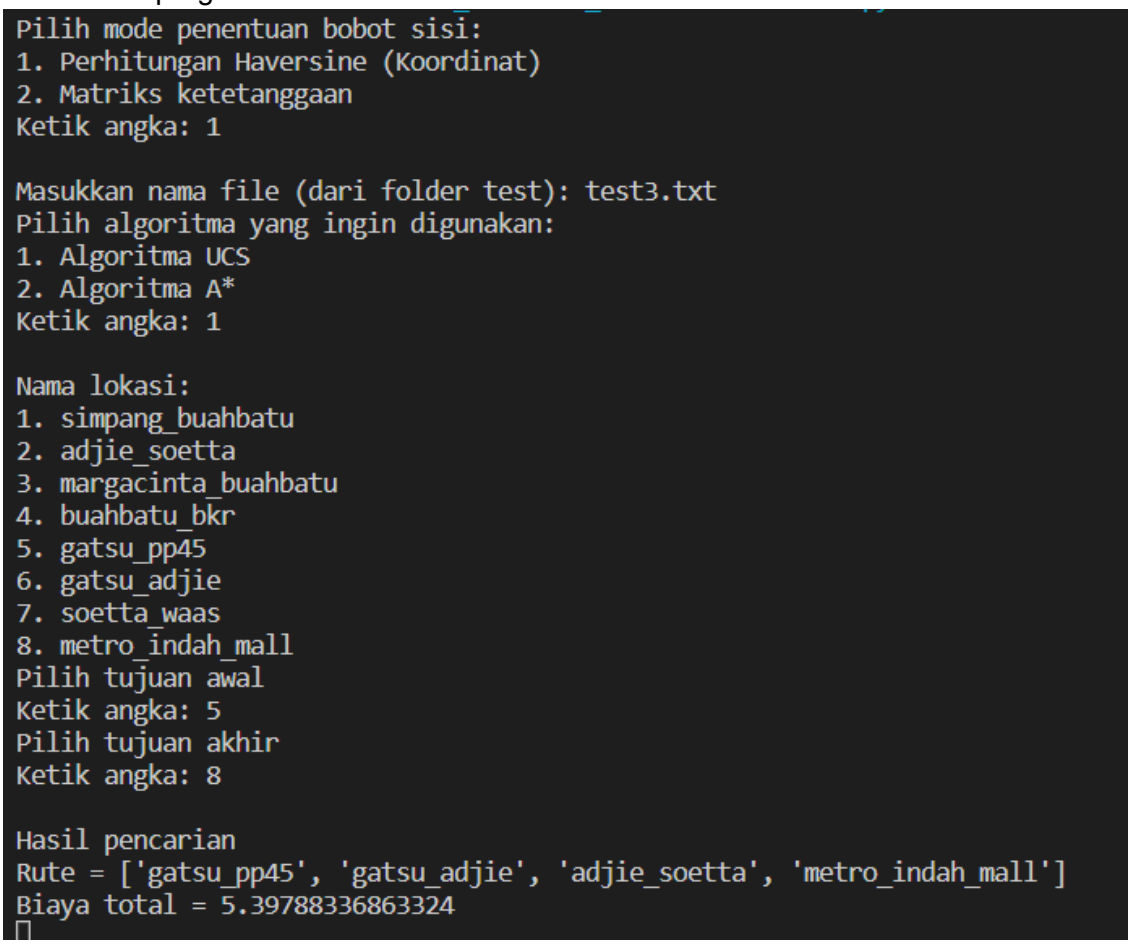
Konfigurasi file (bobot sisi = Haversine)

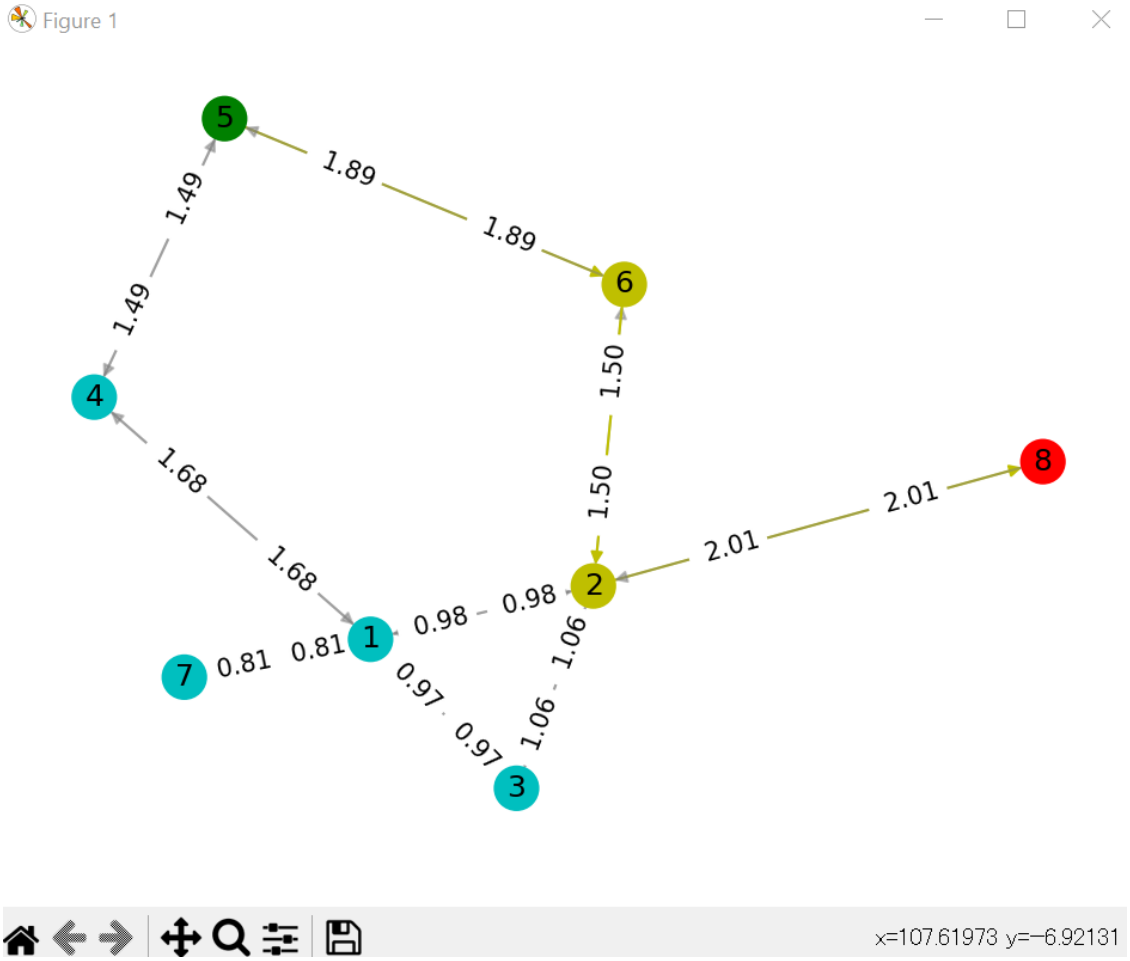
```
simpang_buahbatu -6.94779 107.63328
adjie_soetta -6.94541 107.64186
margacinta_buahbatu -6.95444 107.6389
buahbatu_bkr -6.93698 107.62264
gatsu_pp45 -6.92455 107.62766
gatsu_adjie -6.93195 107.64305
soetta_waas -6.94948 107.62611
metro_indah_mall -6.93986 107.65917
~
0 1 1 1 0 0 1 0
1 0 1 0 0 1 0 1
1 1 0 0 0 0 0 0
1 0 0 0 1 0 0 0
0 0 0 1 0 1 0 0
0 1 0 0 1 0 0 0
1 0 0 0 0 0 0 0
```

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

Algoritma UCS (awal = “gatsu_pp45”, akhir = “metro_indah_mall”)

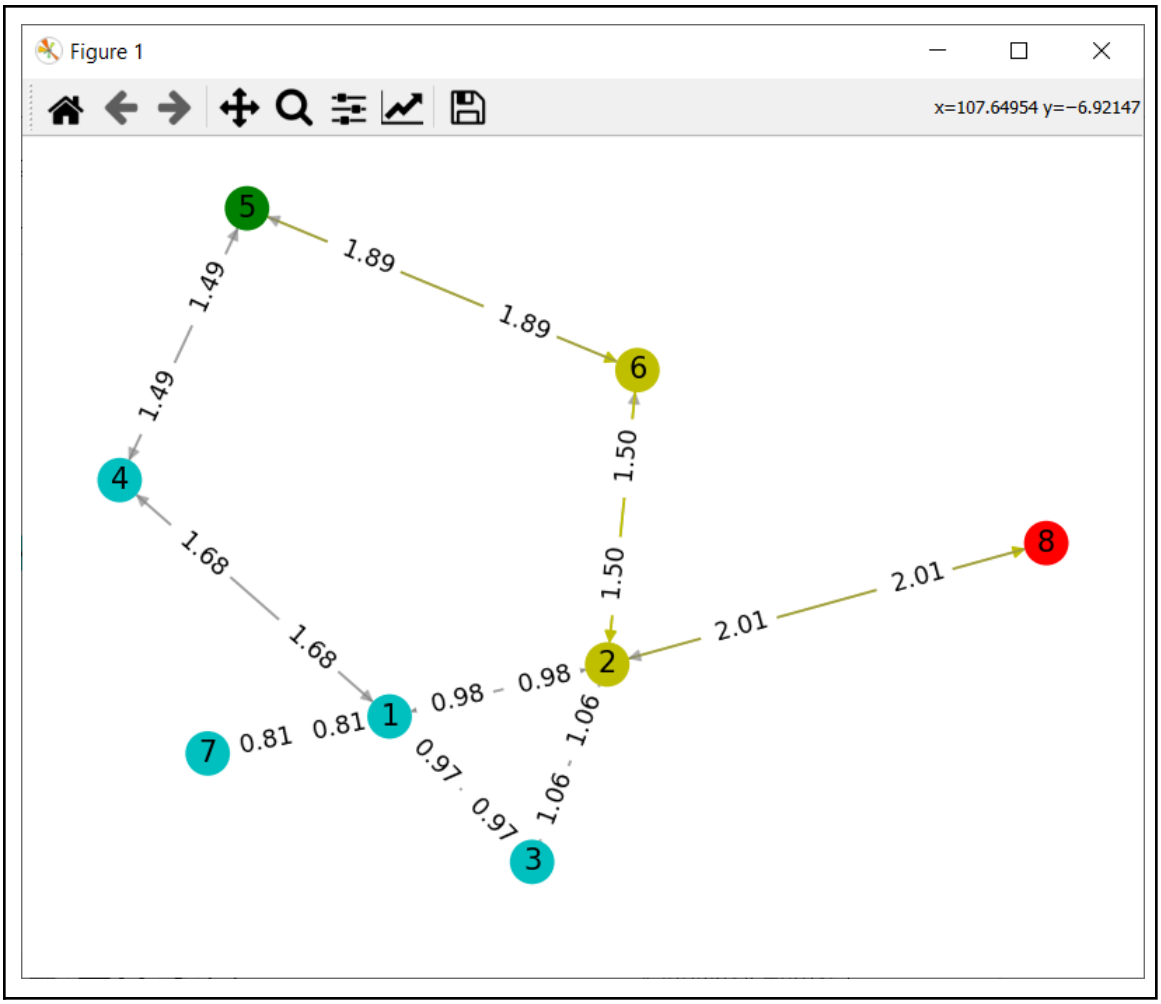
Screenshot program dan visualisasi





Algoritma A* (awal = “gatsu_pp45”, akhir = “metro_indah_mall”)
Screenshot program dan visualisasi

```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1

Masukkan nama file (dari folder test): test3.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 2

Nama lokasi:
1. simpang_buahbatu
2. adjie_soetta
3. margacinta_buahbatu
4. buahbatu_bkr
5. gatsu_pp45
6. gatsu_adjie
7. soetta_waas
8. metro_indah_mall
Pilih tujuan awal
Ketik angka: 5
Pilih tujuan akhir
Ketik angka: 8

Hasil pencarian
Rute = ['gatsu_pp45', 'gatsu_adjie', 'adjie_soetta', 'metro_indah_mall']
Biaya total = 5.39788336863324
```



4.4. test4.txt

Konfigurasi file (bobot sisi = Matriks)

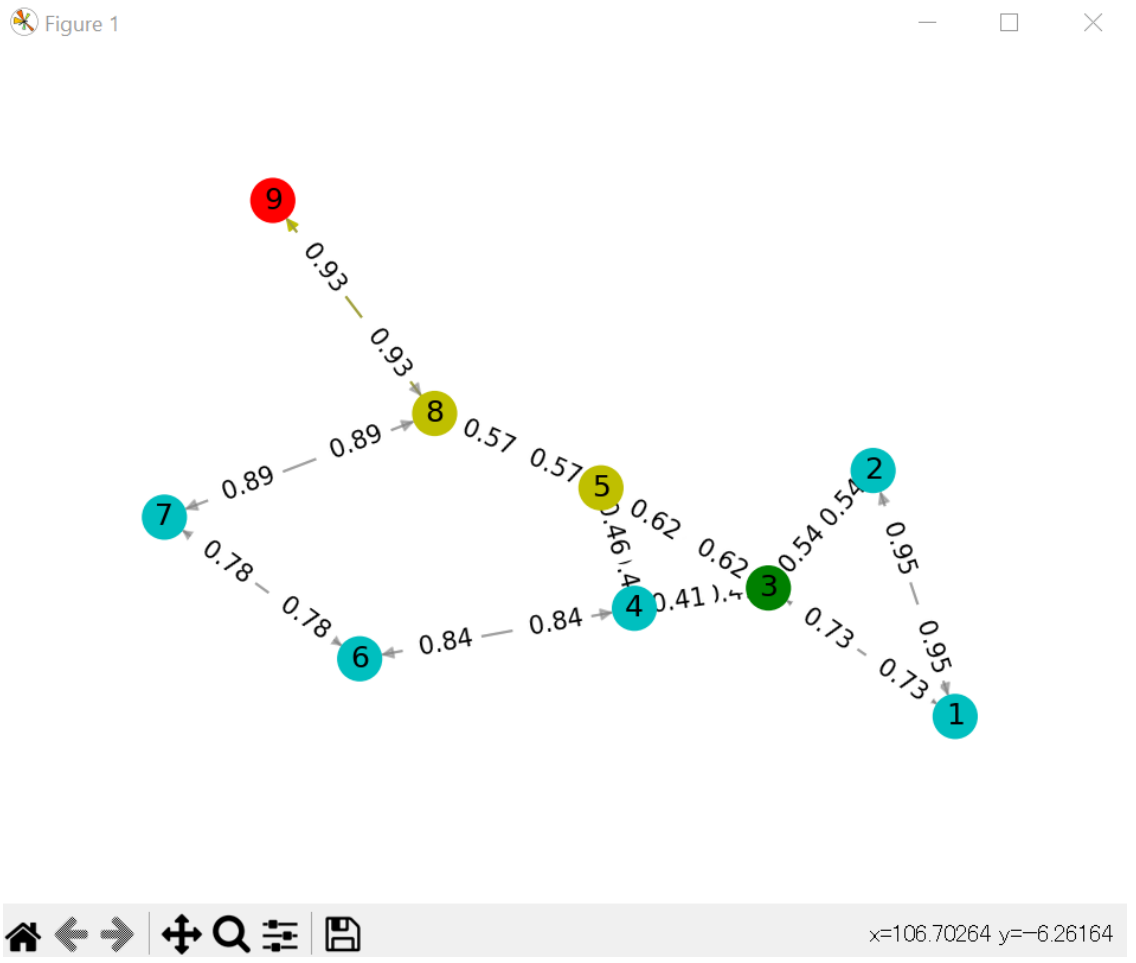
```
talaga_sampireun -6.28223 106.72794
lotte -6.274 106.72573
permata -6.27793 106.72291
pasmod -6.27861 106.7193
giant -6.27458 106.7184
kasturi -6.2803 106.7119
grahataman -6.27556 106.70664
haka -6.27209 106.71392
kebayoran -6.26496 106.70956
~
0 0.9471732000437725 0.7332810293062128 0 0 0 0 0 0
0.9471732000437725 0 0.5367648344654303 0 0 0 0 0 0
0.7332810293062128 0.5367648344654303 0 0.4061073887834696
```

```
0.6222894469154036 0 0 0 0
0 0 0.4061073887834696 0 0.4590238941990119 0.8392158725755317
0 0 0
0 0 0.6222894469154036 0.4590238941990119 0 0 0
0.5673214110976137 0
0 0 0 0.8392158725755317 0 0 0.7847270916428706 0 0
0 0 0 0 0 0.7847270916428706 0 0.8923791859303384 0
0 0 0 0 0.5673214110976137 0 0.8923791859303384 0
0.9277940099972355
0 0 0 0 0 0 0 0.9277940099972355 0
```

Algoritma UCS (awal = “permata”, akhir = “kebayoran”)

Screenshot program dan visualisasi

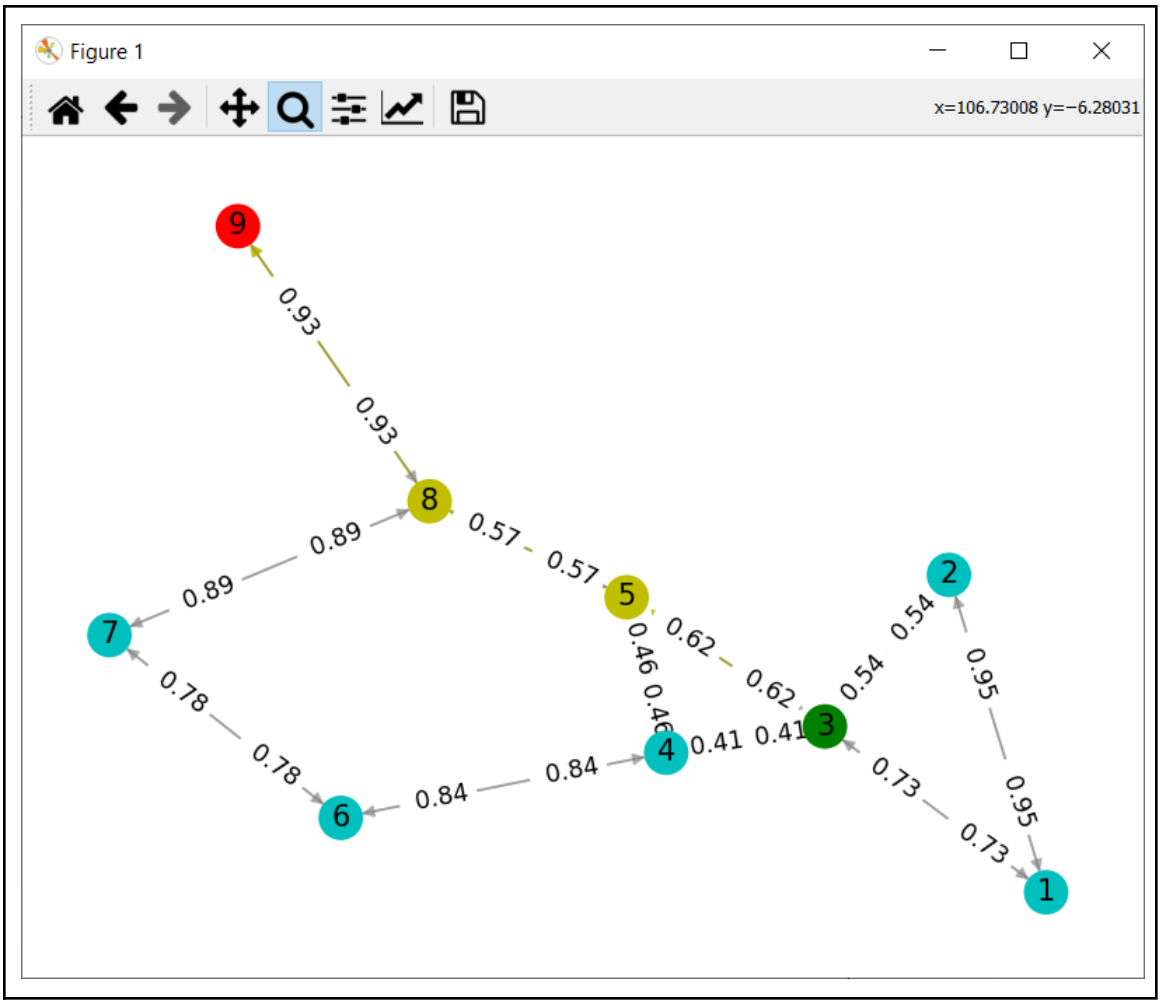
```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 2

Masukkan nama file (dari folder test): test4.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 1

Nama lokasi:
1. talaga_sampireun
2. lotte
3. permata
4. pasmod
5. giant
6. kasturi
7. grahataman
8. haka
9. kebayoran
Pilih tujuan awal
Ketik angka: 3
Pilih tujuan akhir
Ketik angka: 9

Hasil pencarian
Rute = ['permata', 'giant', 'haka', 'kebayoran']
Biaya total = 2.1174048680102526
```



Algoritma A* (awal = “permata”, akhir = “kebayoran”)

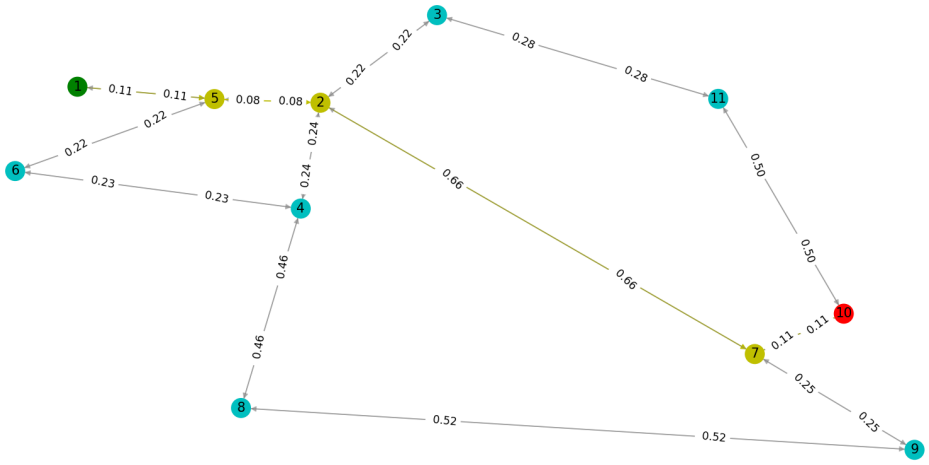
Screenshot program dan visualisasi

```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 2

Masukkan nama file (dari folder test): test4.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 2

Nama lokasi:
1. talaga_sampireun
2. lotte
3. permata
4. pasmod
5. giant
6. kasturi
7. grahataman
8. haka
9. kebayoran
Pilih tujuan awal
Ketik angka: 3
Pilih tujuan akhir
Ketik angka: 9

Hasil pencarian
Rute = ['permata', 'giant', 'haka', 'kebayoran']
Biaya total = 2.1174048680102526
```

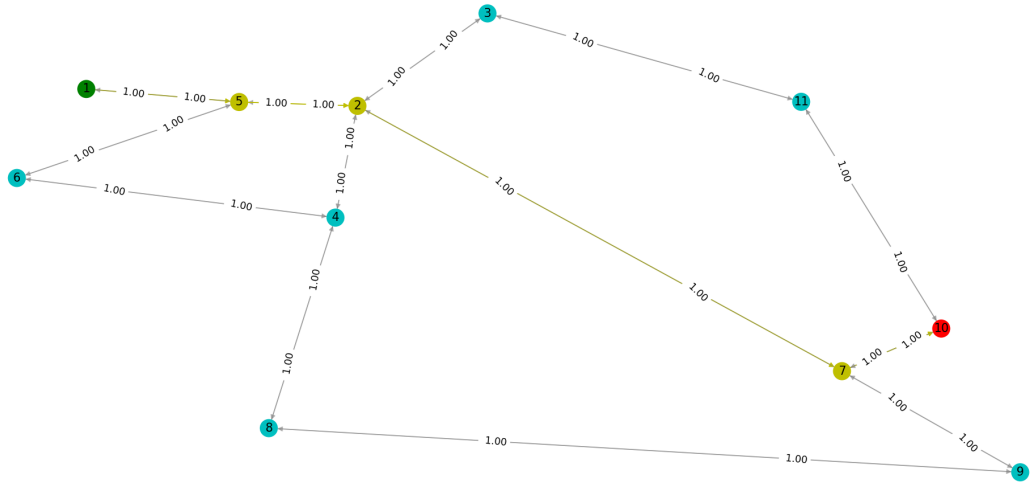


4.5. Keterangan Tambahan

- Graf yang dibaca oleh program berupa graf berarah yang berbobot. Untuk penentuan bobot sisi, pengguna dapat memilih untuk menghitung bobot sisi berdasarkan koordinat atau berdasarkan angka di matriks. Perbedaannya dapat dilihat pada bobot sisi pada kedua gambar di bawah.



Gambar 4.5.1 Pengujian test1.txt jika Menggunakan Jarak Haversine sebagai Bobot Sisi



Gambar 4.5.2 Pengujian test1.txt jika Menggunakan Matriks sebagai Bobot Sisi

- Penomoran simpul pada hasil visualisasi mengikuti urutan pencetakan daftar simpul pada CLI.
- Pewarnaan simpul pada hasil visualisasi mengikuti ketentuan berikut.
  - Simpul hijau adalah simpul awal pencarian rute.
  - Simpul kuning dan sisi kuning adalah simpul dan sisi yang dilalui dalam rute solusi.
  - Simpul merah adalah simpul akhir pencarian rute.
  - Warna cyan dan abu-abu pada simpul dan sisi lain menunjukkan bahwa simpul tersebut bukan bagian dari rute solusi.
- Program mungkin tidak menghasilkan solusi apabila terdapat simpul yang terpisah atau hanya terhubung searah. Sebagai contoh, misal matriks pada test3.txt diubah menjadi sebagai berikut.

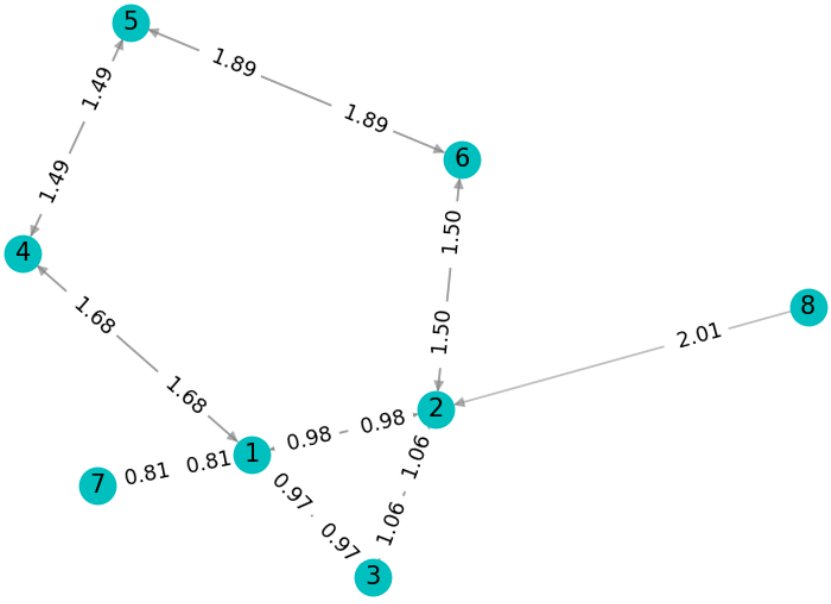
```
0 1 1 1 0 0 1 0
1 0 1 0 0 1 0 0
```

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

Jika dilakukan pencarian dari simpul-1 ke simpul-8, program tidak akan mendapatkan rute, dan ditampilkan sebagai berikut.

```
Pilih mode penentuan bobot sisi:
1. Perhitungan Haversine (Koordinat)
2. Matriks ketetanggaan
Ketik angka: 1

Masukkan nama file (dari folder test): test3.txt
Pilih algoritma yang ingin digunakan:
1. Algoritma UCS
2. Algoritma A*
Ketik angka: 1

Nama lokasi:
1. simpang_buahbatu
2. adjie_soetta
3. margacinta_buahbatu
4. buahbatu_bkr
5. gatsu_pp45
6. gatsu_adjie
7. soetta_waas
8. metro_indah_mall
Pilih tujuan awal
Ketik angka: 1
Pilih tujuan akhir
Ketik angka: 8

Hasil pencarian
Rute tidak ditemukan
```

Gambar 4.5.3 Contoh Pengujian Tanpa Solusi



Gambar 4.5.4 Contoh Visualisasi Tanpa Solusi

4.6. Analisis

Pada keempat pengujian di atas, algoritma UCS dan A* search menghasilkan solusi yang sama. Apabila diteliti, solusi dari kedua algoritma adalah solusi optimal dari graf.

Meskipun kedua algoritma menghasilkan solusi yang sama, hal yang membedakan keduanya adalah efisiensi proses pencarian. Dengan mengaktifkan mode *debug* pada program utama (file main.py, baris 46, kolom 71, pada parameter `show_debug`), pencarian menggunakan algoritma A* tampak lebih efisien dari pada algoritma UCS. Hal ini karena pembangkitan simpul tetangga pada algoritma A* lebih terarah oleh akibat adanya nilai heuristik, sehingga jumlah simpul yang perlu ditelusuri lebih sedikit.

```
Hasil pencarian
0 --- sekeloa_tubisdalam -- 0 ['sekeloa_tubisdalam']
0.11371713600111909 --- dipatiukur_sekeloa -- 0.11371713600111909 ['sekeloa_tubisdalam', 'dipatiukur_sek
eloa']
0.3628550878806914 --- teukuumar_dipatiukur -- 0.3628550878806914 ['sekeloa_tubisdalam', 'dipatiukur_sek
eloa', 'teukuumar_dipatiukur']
0.4984971205667796 --- tubisluar_tubisdalam -- 0.4984971205667796 ['sekeloa_tubisdalam', 'tubisluar_tubi
sdalam']
0.7729297998109118 --- simpang_dago -- 0.7729297998109118 ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', '
simpang_dago']
0.7832775507515659 --- tubisluar_djuanda -- 0.7832775507515659 ['sekeloa_tubisdalam', 'tubisluar_tubisda
lam', 'tubisluar_djuanda']
0.8530686763208668 --- siliwangi_sumurbandung -- 0.8530686763208668 ['sekeloa_tubisdalam', 'dipatiukur_s
ekeloa', 'simpang_dago', 'siliwangi_sumurbandung']
0.8781095076748078 --- djuanda_teukuumar -- 0.8781095076748078 ['sekeloa_tubisdalam', 'dipatiukur_sekelo
a', 'teukuumar_dipatiukur', 'djuanda_teukuumar']
0.9598383051668624 --- sangkuriang_siliwangi -- 0.9598383051668624 ['sekeloa_tubisdalam', 'dipatiukur_se
keloa', 'simpang_dago', 'siliwangi_sumurbandung', 'sangkuriang_siliwangi']
Rute = ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', 'simpang_dago', 'siliwangi_sumurbandung', 'sangkuria
ng_siliwangi']
Biaya total = 0.9598383051668624
```

Gambar 4.6.1 Urutan Pencarian Pengujian test1.txt (UCS)

```
Hasil pencarian
0.7742080961931881 --- sekeloa_tubisdalam -- 0 ['sekeloa_tubisdalam']
0.9076693020614313 --- dipatiukur_sekeloa -- 0.11371713600111909 ['sekeloa_tubisdalam', 'dipatiukur_seke
loa']
0.9592351354643257 --- simpang_dago -- 0.7729297998109118 ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', '
simpang_dago']
0.9598383051668624 --- siliwangi_sumurbandung -- 0.8530686763208668 ['sekeloa_tubisdalam', 'dipatiukur_s
ekeloa', 'simpang_dago', 'siliwangi_sumurbandung']
0.9598383051668624 --- sangkuriang_siliwangi -- 0.9598383051668624 ['sekeloa_tubisdalam', 'dipatiukur_se
keloa', 'simpang_dago', 'siliwangi_sumurbandung', 'sangkuriang_siliwangi']
Rute = ['sekeloa_tubisdalam', 'dipatiukur_sekeloa', 'simpang_dago', 'siliwangi_sumurbandung', 'sangkuria
ng_siliwangi']
Biaya total = 0.9598383051668624
```

Gambar 4.6.2 Urutan Pencarian Pengujian test1.txt (A)*

```
Hasil pencarian
0 --- permata -- 0 ['permata']
0.4061073887834696 --- pasmod -- 0.4061073887834696 ['permata', 'pasmod']
0.5367648344654303 --- lotte -- 0.5367648344654303 ['permata', 'lotte']
0.6222894469154036 --- giant -- 0.6222894469154036 ['permata', 'giant']
0.7332810293062128 --- talaga_sampireun -- 0.7332810293062128 ['permata', 'talaga_sampireun']
0.8651312829824815 --- giant -- 0.8651312829824815 ['permata', 'pasmod', 'giant']
1.1896108580130171 --- haka -- 1.1896108580130171 ['permata', 'giant', 'haka']
1.2453232613590013 --- kasturi -- 1.2453232613590013 ['permata', 'pasmod', 'kasturi']
1.4324526940800952 --- haka -- 1.4324526940800952 ['permata', 'pasmod', 'giant', 'haka']
1.4839380345092028 --- talaga_sampireun -- 1.4839380345092028 ['permata', 'lotte', 'talaga_sampireun']
2.030050353001872 --- grahataman -- 2.030050353001872 ['permata', 'pasmod', 'kasturi', 'grahataman']
2.0819900439433554 --- grahataman -- 2.0819900439433554 ['permata', 'giant', 'haka', 'grahataman']
2.1174048680102526 --- kebayoran -- 2.1174048680102526 ['permata', 'giant', 'haka', 'kebayoran']
Rute = ['permata', 'giant', 'haka', 'kebayoran']
Biaya total = 2.1174048680102526
```

Gambar 4.6.3 Urutan Pencarian Pengujian test4.txt (UCS)

```
Hasil pencarian
2.0633076165758206 --- permata -- 0 ['permata']
2.0710615119398637 --- giant -- 0.6222894469154036 ['permata', 'giant']
2.1174048680102526 --- haka -- 1.1896108580130171 ['permata', 'giant', 'haka']
2.1174048680102526 --- kebayoran -- 2.1174048680102526 ['permata', 'giant', 'haka', 'kebayoran']
Rute = ['permata', 'giant', 'haka', 'kebayoran']
Biaya total = 2.1174048680102526
```

Gambar 4.6.4 Urutan Pencarian Pengujian test4.txt (A)*

Keterangan: Langkah pencarian dicetak dengan format “<prioritas> --- <nama> --- <total biaya rute> --- <rute yang telah ditempuh>”

BAB 5
PENUTUP

5.1. Kesimpulan

Dari tugas kecil 3 ini, kami memperoleh kesimpulan bahwa kami dapat menemukan rute terpendek antara dua simpul dengan menggunakan algoritma UCS dan A*. Pencarian dengan kedua algoritma bisa meraih solusi yang optimal, tetapi karena pencarian yang lebih terarah dengan memperhitungkan fungsi heuristik h(n), algoritma A* cenderung memiliki performa yang lebih efisien dibandingkan algoritma UCS.

5.2. Saran

Algoritma pencarian lintasan dapat dikembangkan lebih jauh lagi agar pengumpulan data dapat menjadi lebih praktis dengan menaruh *pinpoint* yang menandai sebuah simpul graf dari peta, misal Google Map. Performa program ini masih bisa ditingkatkan lagi jika kami mengintegrasikan program ini dengan bahasa yang lain.

5.3. Komentar dan Refleksi

Kami senang karena pelajaran yang diberikan dosen di kelas (Bu Ulfa) dapat kami implementasikan dengan baik pada tugas kali ini. Kami berterima kasih kepada Bu Ulfa sebagai dosen pengampu kelas K2 serta Pak Rinaldi yang memberikan materi secara lengkap dan jelas.

5.4. Tabel *Checkpoint*

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	

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil3-Stima-2023.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/yuujin-Q/Tucil3_13521048_13521074

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13521048	M Farrel Danendra Rachim	Pembacaan file graf, algoritma A*, testcase, laporan
13521074	Eugene Yap Jin Quan	Setup, graf dan visualisasi graf, algoritma UCS, laporan