LAPORAN TUGAS KECIL

IF2211 STRATEGI ALGORITMA

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun oleh

Noel Christoffel Simbolon	13521096
Zidane Firzatullah	13521163

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKOLOGI BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

1.1 Deskripsi Tugas

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, anda 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. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antar 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.

Jalan Terusan Pasirkoja
Jalan Pasirkoja
Jalan Pagarsih
Jalan Astana Anyar
Jalan Otto Iskandardinata
Jalan Asia Afrika
Jalan Kyai Haji Wahid Hasyim
Jalan Lingkar Selatan
Jalan Peta
Jalan BKR
Jalan Mohammad Toha
Jalan Pasundan
Jalan Lengkong Besar
Jalan Pungkur
Jalan Buah Batu
Warung Muncang
Sukahaji
Babakan Tarogong
Bojongloa Kaler
Suka Asih
Makam Nyengseret - Suka Asih 2
Babakan Asih
Panjunan
Karanganyar
Ramayana Robinson
Alun-alun
Cikapundung
Nyengsret
Puskesmas Pasundan
Balonggede
Pungkur
Universitas Langlangbuana
Ciateul
Taman Tegallega
Situsaeur
Ciroyom
Astana Anyar
Pelindung Hewan
Kopo
Jalan Muara
Jalan Kembar Baru
Jalan Pelindung Hewan
Babakan Ciparay
Mollis
Jamika
Jalan Cibadak
Jalan Dalem Kaum
Jalan Lengkong Kecil
Jalan Paledang
Jalan Cikawao
Jalan Madesa
Jalan Pabaki
Jalan Kalipah Apo

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, lalu program 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.

1.2 Spesifikasi Program

Terdapat beberapa spesifikasi dari program yang akan diimplementasikan:

1. Program menerima input *file* graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.

2. Program dapat menampilkan peta/graf.
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

BAB II
IMPLEMENTASI PROGRAM

2.1 `main_algorithm.py`

```python
import heapq

from src.common import distance_operations
from src.model.node import Node

class Engine:
    @staticmethod
    def __heuristic(informed: bool, start_node: Node = None, goal_node: Node =
None) -> float:
        if not informed:
            return 0
        return distance_operations.euclidean_distance(start_node, goal_node)

    @staticmethod
    def __trace_path(dijkstra_table: list[(float, int)], node_list: list[Node],
                     goal_index: int) -> (float, list[Node]):
        current_index = goal_index
        path: list[Node] = []
        cost: float = dijkstra_table[goal_index][0]

        while current_index != -1:
            path.append(node_list[current_index])
            current_index = dijkstra_table[current_index][1]

        path.reverse()
        return cost, path

    @staticmethod
    def search_path(start_index: int, goal_index: int, informed: bool,
                    node_list: list[Node], adj_matrix: list[list[float]]) ->
(float, list[Node]):
        if start_index == goal_index:
            return adj_matrix[start_index][goal_index], [node_list[start_index]]

        # format: (shortest distance, source node id)
        dijkstra_table = [(-1.0, 0) for _ in range(len(node_list))]
        dijkstra_table[start_index] = (Engine.__heuristic(informed,
node_list[start_index], node_list[goal_index]), -1)

        queue = []
        # format: (shortest distance, node)
        heapq.heappush(queue,
                       (Engine.__heuristic(informed, node_list[start_index],
node_list[goal_index]), start_index))

        while len(queue) > 0:
            node_index: int = heapq.heappop(queue)[1]
            current_node: Node = node_list[node_index]

            # found goal
            if current_node == node_list[goal_index]:
                return Engine.__trace_path(dijkstra_table, node_list, goal_index)
            for i, weight in enumerate(adj_matrix[node_index]):
                # just to make sure :)
                if -0.0001 <= adj_matrix[node_index][i] <= 0.0001:
                    continue

                node = node_list[i]

                weight = dijkstra_table[current_node.node_id][0]
                weight += adj_matrix[node_index][i]
                weight += Engine.__heuristic(informed, node,
node_list[goal_index])
                weight -= Engine.__heuristic(informed, current_node,
node_list[goal_index])

                if dijkstra_table[node.node_id][0] > weight or
dijkstra_table[node.node_id][0] == -1:
                    dijkstra_table[node.node_id] = (weight, current_node.node_id)
                    heapq.heappush(queue, (weight, node.node_id))

        return None

    @staticmethod
    def search_astar(start_index: int, goal_index: int, node_list: list[Node],
                     adj_matrix: list[list[float]]) -> (float, list[Node]):
        return Engine.search_path(start_index, goal_index, True, node_list,
adj_matrix)

    @staticmethod
    def search_ucs(start_index: int, goal_index: int, node_list: list[Node],
                   adj_matrix: list[list[float]]) -> (float, list[Node]):
        return Engine.search_path(start_index, goal_index, False, node_list,
adj_matrix)
```

2.2 distance_operations.py

```
import math

from geopy.distance import distance

from src.model.node import Node

def euclidean_distance(first_node: Node, second_node: Node) -> float:
    """
    :param first_node: the first node
    :param second_node: the second node
    :return: the Euclidean distance between the two nodes
    """
    return math.sqrt(math.pow(first_node.y - second_node.y, 2) +
math.pow(first_node.x - second_node.x, 2))

def geodesic_distance(first_coords: tuple[float, float], second_coords:
tuple[float, float]) -> float:
    """
    :param first_coords: the coordinates of the first location on Earth's surface
    :param second_coords: the coordinates of the second location on Earth's
surface
```

```
    :return: the geodesic distance between the two locations (in meters)
    """
    return distance(first_coords, second_coords).meters
```

2.3 file_tab.py

```
import customtkinter
import matplotlib.pyplot as plt
import networkx as nx
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk

from src.algorithm.main_algorithm import Engine
from src.gui import util
from src.io.file_handler import FileInputHandler
from src.model.node import Node

class FileTab(customtkinter.CTkFrame):
    # Algorithm arguments
    starting_index: int = 0
    destination_index: int = 0
    nodes: list[Node] = []  # List of available Nodes
    adj_matrix: list[list[float]] = [[]]  # Two-dimensional list to hold the
adjacency matrix

    # Output
    route: list[Node] = []  # List of route Nodes
    distance: float = 0

    # Attribute to save node positions in the graph visualization
    input_graph: nx.Graph = None
    node_positions: dict = {}

    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        button_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='bold')

        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=0)
        self.grid_columnconfigure(1, weight=1)

        self.file_input_frame = FileInputFrame(master=self, width=180)
        self.file_input_frame.grid(row=0, column=0, padx=(10, 5), pady=10,
sticky='nswe')

        # Open file button
        self.open_file_button =
customtkinter.CTkButton(master=self.file_input_frame,
                                                        text='Open File',
                                                        font=button_font,
                                                        command=self.parse_file_a
nd_visualize_input_graph)
        self.open_file_button.grid(row=1, column=0, padx=20, pady=(10, 0))

        # Start button
        self.start_button = customtkinter.CTkButton(master=self.file_input_frame,
                                                    text='Start',
                                                    font=button_font,
                                                    command=self.find_shortest_ro
ute_and_visualize_route)
        self.start_button.grid(row=9, column=0, padx=20, pady=(10, 0))

        self.file_output_frame = FileOutputFrame(master=self)
        self.file_output_frame.grid(row=0, column=1, padx=(5, 10), pady=10,
sticky='nswe')

    def parse_file_and_visualize_input_graph(self) -> None:
        # Clear any text output first
        self.file_input_frame.file_validation_message.configure(text='')
        self.file_input_frame.status_message.configure(text='')

        try:
            FileTab.parse_file()
        except FileNotFoundError:
            self.file_input_frame.file_validation_message.configure(text='Please
open a file.',
                                                                    text_color='r
ed')
            return
        except RuntimeError:
            self.file_input_frame.file_validation_message.configure(text='Invalid
file.',
                                                                    text_color='r
ed')
            return

        if len(FileTab.nodes) < 8:
            self.file_input_frame.file_validation_message.configure(text='Minimum
8 nodes required.',
                                                                    text_color='r
ed')
            return

        self.visualize_input_graph()

    @staticmethod
    def parse_file() -> None:
        file_path = customtkinter.filedialog.askopenfilename(title='Open a Text
File',
                                                             filetypes=[("Text
Files", "*.txt")])
        nodes, adj_matrix = FileInputHandler.load_file(file_path)

        FileTab.nodes = nodes
        FileTab.adj_matrix = adj_matrix

    def visualize_input_graph(self) -> None:
        num_nodes = len(FileTab.adj_matrix)

        # Create an empty graph
        FileTab.input_graph = nx.DiGraph()

        # Add nodes to the graph
        FileTab.input_graph.add_nodes_from([node.node_id for node in
FileTab.nodes])

        # Add edges to the graph based on the adjacency matrix
        for i in range(num_nodes):
            for j in range(num_nodes):
                weight = FileTab.adj_matrix[i][j]
                if weight > 0:
                    FileTab.input_graph.add_edge(i, j, weight=weight)  # Add edge
with weight as an attribute

        # Compute the spring layout for node positions
        FileTab.node_positions = nx.spring_layout(FileTab.input_graph)

        # Create a Figure object
        fig = plt.figure(figsize=(7, 4))  # Width and height in

        # Configure subplot params
        plt.subplots_adjust(left=0, bottom=0, right=1, top=1)

        # Remove black borders
        ax = fig.gca()
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.spines['bottom'].set_visible(False)
        ax.spines['left'].set_visible(False)

        # Draw nodes
        nx.draw_networkx_nodes(FileTab.input_graph, pos=FileTab.node_positions)

        # Draw edges
        nx.draw_networkx_edges(FileTab.input_graph, pos=FileTab.node_positions)

        # Draw node labels
        nx.draw_networkx_labels(FileTab.input_graph, pos=FileTab.node_positions)

        # Destroy graph canvas and navigation toolbar if present
        self.file_output_frame.graph_canvas.get_tk_widget().destroy()
        self.file_output_frame.graph_navigation_toolbar.destroy()

        # Visualize the graph
        self.file_output_frame.graph_canvas = FigureCanvasTkAgg(fig,
master=self.file_output_frame.graph_frame)
        self.file_output_frame.graph_canvas.draw()
        self.file_output_frame.graph_canvas.get_tk_widget().pack()

        # Create a navigation toolbar for the graph
        self.file_output_frame.graph_navigation_toolbar =
NavigationToolbar2Tk(self.file_output_frame.graph_canvas)
        self.file_output_frame.graph_navigation_toolbar.update()
        self.file_output_frame.graph_canvas.get_tk_widget().pack()

        self.file_input_frame.file_validation_message.configure(text='Graph
visualized.', text_color='green')

    def find_shortest_route_and_visualize_route(self) -> None:
        # If the user hasn't opened a file
        if len(FileTab.nodes) == 0:
            # Clear status message
            self.file_input_frame.status_message.configure(text="")
            self.file_input_frame.status_message.configure(text="Please open a
file first.",
                                                           text_color='red')
            return

        # If the starting node index or the destination node index is not valid,
        # display error message until it is valid
        if not
(FileTab.is_index_valid(self.file_input_frame.starting_node_entry.get()) and
                FileTab.is_index_valid(self.file_input_frame.destination_node_ent
ry.get())):

            # Clear status message
            self.file_input_frame.status_message.configure(text="")

            # If the starting node index is not valid
            if not
FileTab.is_index_valid(self.file_input_frame.starting_node_entry.get()):
                self.file_input_frame.status_message.configure(text="Invalid
starting node.",
                                                               text_color='red')
                return
            else:
                self.file_input_frame.status_message.configure(text="")

            # If the destination node index is not valid
            if not
FileTab.is_index_valid(self.file_input_frame.destination_node_entry.get()):
                self.file_input_frame.status_message.configure(text="Invalid
destination node.",
                                                               text_color='red')
                return
            else:
                self.file_input_frame.status_message.configure(text="")

        try:
            self.find_shortest_route()
        except TypeError:
            # There is no route from the starting node to the destination node
            self.file_input_frame.status_message.configure(text="No route
found.",
                                                           text_color='red')
            return

        self.visualize_route()
        self.show_route_and_distance()

    def find_shortest_route(self) -> None:
        FileTab.starting_index =
int(self.file_input_frame.starting_node_entry.get())
        FileTab.destination_index =
int(self.file_input_frame.destination_node_entry.get())

        # Start algorithm
        if self.file_input_frame.algorithm_options.get() == 'A*':
            # A-star path-finding
            FileTab.distance, FileTab.route =
Engine.search_astar(FileTab.starting_index,
                                                                   FileTab.destina
tion_index,
                                                                   FileTab.nodes,
                                                                   FileTab.adj_mat
rix)
        else:
            # UCS path-finding
            FileTab.distance, FileTab.route =
Engine.search_ucs(FileTab.starting_index,
                                                                 FileTab.destinati
on_index,
                                                                 FileTab.nodes,
                                                                 FileTab.adj_matri
x)

    def visualize_route(self) -> None:
        # Create a Figure object
        fig = plt.figure(figsize=(7, 4))

        # Configure subplot params
        plt.subplots_adjust(left=0, bottom=0, right=1, top=1)

        # List of node IDs to color edges between
        route_nodes = [node.node_id for node in FileTab.route]

        # Iterate over edges to get the edges that needs to be colored yellow
        route_edges = []
        i = 0
        while i < len(route_nodes) - 1:
            for edge in FileTab.input_graph.edges():
                if i + 1 < len(route_nodes):
                    if (edge[0] == route_nodes[i] and edge[1] == route_nodes[i +
1]) or \
                            (edge[1] == route_nodes[i] and edge[0] ==
route_nodes[i + 1]):
                        route_edges.append(edge)
                        i += 1
                else:
                    break

        # Color the route edges yellow
        for edge in FileTab.input_graph.edges():
            if edge in route_edges:
                FileTab.input_graph.edges[edge]['edge_color'] = 'yellow'
            else:
                FileTab.input_graph.edges[edge]['edge_color'] = 'black'

        # Draw nodes
        nx.draw_networkx_nodes(FileTab.input_graph, pos=FileTab.node_positions)

        # Draw edges
        nx.draw_networkx_edges(FileTab.input_graph, pos=FileTab.node_positions)

        # Draw node labels
        nx.draw_networkx_labels(FileTab.input_graph, pos=FileTab.node_positions)
        # Draw the graph with edge colors
        edge_colors = [FileTab.input_graph.edges[edge]['edge_color'] for edge in
FileTab.input_graph.edges()]
        nx.draw(FileTab.input_graph, pos=FileTab.node_positions,
with_labels=True, edge_color=edge_colors)

        # Destroy graph canvas (if a plot is present this destroys it)
        self.file_output_frame.graph_canvas.get_tk_widget().destroy()

        # Visualize the graph
        self.file_output_frame.graph_canvas = FigureCanvasTkAgg(fig,
                                                                master=self.file_
output_frame.graph_frame)
        self.file_output_frame.graph_canvas.draw()
        self.file_output_frame.graph_canvas.get_tk_widget().pack()

        self.file_input_frame.status_message.configure(text='Route visualized.',
                                                       text_color='green')

    def show_route_and_distance(self) -> None:
        # Convert the route to string, example: 2-1-6-3-7
        route = ''
        for route_node in FileTab.route:
            route += f'{route_node.node_id}-'
        route = route[:-1]  # Remove trailing hyphen

        self.file_output_frame.result_tab_view.route_label.configure(text=route)
        self.file_output_frame.result_tab_view.distance_label.configure(text=f'{F
ileTab.distance:.5f} m')

    @staticmethod
    def is_index_valid(index: int) -> bool:
        """
        Validates the node index
        :param index: node's index
        :return: True if the index is valid, False otherwise
        """
        try:
            if 0 <= int(index) <= len(FileTab.adj_matrix) - 1:
                return True
        except ValueError:
            return False

        return False

class FileInputFrame(customtkinter.CTkFrame):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Custom-defined fonts
        title_font = customtkinter.CTkFont(family='Segoe UI', size=-18,
weight='bold')
        message_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
        input_label_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='bold')
        entry_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
```

```
        select_theme_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
        select_algorithm_font = customtkinter.CTkFont(family='Segoe UI', size=-
13, weight='normal')

        # Expandable empty space between the Start button and UI theme options
        self.grid_rowconfigure(11, weight=1)

        # App title
        self.app_title = customtkinter.CTkLabel(master=self,
                                                text='Compass',
                                                font=title_font)
        self.app_title.grid(row=0, column=0, padx=20, pady=(20, 10))

        # Input file button is defined in FileTab

        self.file_validation_message = customtkinter.CTkLabel(master=self,
                                                              text='',
                                                              font=message_font)
        self.file_validation_message.grid(row=2, column=0, padx=20, pady=0)

        # Starting node and destination node
        self.starting_node_label = customtkinter.CTkLabel(master=self,
                                                          text='Starting node:',
                                                          font=input_label_font)
        self.starting_node_label.grid(row=3, column=0, padx=20, pady=0)

        self.starting_node_entry = customtkinter.CTkEntry(master=self,
                                                          placeholder_text="Node
number",
                                                          font=entry_font)
        self.starting_node_entry.grid(row=4, column=0, padx=20, pady=0)

        self.destination_node_label = customtkinter.CTkLabel(master=self,
                                                             text='Destination
node:',
                                                             font=input_label_fon
t)
        self.destination_node_label.grid(row=5, column=0, padx=20, pady=0)

        self.destination_node_entry = customtkinter.CTkEntry(master=self,
                                                             placeholder_text="No
de number",
                                                             font=entry_font)
        self.destination_node_entry.grid(row=6, column=0, padx=20, pady=(0, 5))

        # Select algorithm
        self.select_algorithm_label = customtkinter.CTkLabel(master=self,
                                                             text="Select
algorithm:",
                                                             font=input_label_fon
t)
        self.select_algorithm_label.grid(row=7, column=0, padx=20, pady=(5, 0))

        self.algorithm_options = customtkinter.CTkOptionMenu(master=self,
                                                             values=['A*',
'UCS'],
                                                              font=select_algorith
m_font)
        self.algorithm_options.grid(row=8, column=0, padx=20, pady=(0, 10))

        # Start button is defined in FileTab

        self.status_message = customtkinter.CTkLabel(master=self,
                                                     text='',
                                                     font=message_font)
        self.status_message.grid(row=10, column=0, padx=20, pady=0)

        # Select GUI theme
        self.select_theme_label = customtkinter.CTkLabel(master=self,
                                                         text="Select theme:",
                                                         font=select_theme_font)
        self.select_theme_label.grid(row=12, column=0, padx=20, pady=0)

        self.theme_options = customtkinter.CTkOptionMenu(master=self,
                                                         values=["Dark", "Light",
"System"],
                                                         font=select_theme_font,
                                                         command=util.change_appe
arance_mode)
        self.theme_options.grid(row=13, column=0, padx=20, pady=(0, 20))

class FileOutputFrame(customtkinter.CTkFrame):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Configure the grid system
        self.grid_rowconfigure((0, 2), weight=1)
        self.grid_columnconfigure((0, 2), weight=1)
        self.grid_rowconfigure(1, weight=0)
        self.grid_columnconfigure(1, weight=0)
        self.grid_rowconfigure(3, weight=0)

        # Transparent frame to hold the graph visualization
        self.graph_frame = customtkinter.CTkFrame(master=self,
                                                  fg_color='transparent')
        self.graph_frame.grid(row=1, column=1)

        # Initialize a canvas for graph visualization
        # This initialization is useful to avoid displaying multiple plots
        # by destroying the canvas before creating a new one every time a graph
is want to be drawn
        self.graph_canvas = FigureCanvasTkAgg(None, master=self.graph_frame)

        # Initialize a navigation toolbar for graph
        # This initialization is useful to avoid displaying multiple toolbars
        # by destroying it before creating a new one every time a graph is want
to be drawn
        self.graph_navigation_toolbar = NavigationToolbar2Tk(self.graph_canvas)
        self.graph_navigation_toolbar.pack_forget()

        # Result tab view
        self.result_tab_view = FileResultTabView(master=self,
                                                 height=125)
        self.result_tab_view.grid(row=3, column=0, columnspan=3, padx=20,
pady=20, sticky='we')
        self.result_tab_view._segmented_button.configure(font=('Segoe UI', -13,
'bold'))

class FileResultTabView(customtkinter.CTkTabview):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        output_font = customtkinter.CTkFont(family='Segoe UI', size=-14,
weight='normal')

        # Create two tabs for output
        self.add('Route')
        self.add('Distance')

        self.tab('Route').grid_columnconfigure(0, weight=1)
        self.tab('Distance').grid_columnconfigure(0, weight=1)

        # Create a scrollable frame to hold the route
        self.route_scrollable_frame =
customtkinter.CTkScrollableFrame(master=self.tab('Route'),
                                                                        height=30,
                                                                        orientatio
n='horizontal')
        self.route_scrollable_frame.grid(row=0, column=0, columnspan=1, pady=10,
sticky='ns')

        self.route_label =
customtkinter.CTkLabel(master=self.route_scrollable_frame,
                                                  text='',
                                                  text_color='green',
                                                  font=output_font)
        self.route_label.grid(row=0, column=0, padx=10)

        # Create a label for distance
        self.distance_label = customtkinter.CTkLabel(master=self.tab('Distance'),
                                                     text='',
                                                     text_color='green',
                                                     font=output_font)
        self.distance_label.grid(row=0, column=0, pady=(20, 0))
```

2.4 main_window.py

```
import customtkinter

from src.gui import file_tab, map_tab

customtkinter.set_appearance_mode("Dark")
customtkinter.set_default_color_theme("blue")

class MainWindow(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        # Main window configurations
        self.title("Compass")
        self.minsize(1000, 725)

        # Configure grid for the Compass tab view
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)

        # Configure tab view for Compass app
        self.compass_tab_view = CompassTabView(master=self)
        self.compass_tab_view.grid(row=0, column=0, padx=20, pady=20,
sticky='nswe')
        self.compass_tab_view._segmented_button.configure(font=('Segoe UI', -13,
'bold'))

class CompassTabView(customtkinter.CTkTabview):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Create two tabs to handle input from a file and from map
        self.add('Input from File')
        self.add('Input from Map')

        self._tab_dict['Input from File'] = file_tab.FileTab(master=self)
        self._tab_dict['Input from Map'] = map_tab.MapTab(master=self)

        self.set('Input from File')
```

2.5 map_tab.py

```
import customtkinter
import tkintermapview
from tkintermapview.canvas_path import CanvasPath
from tkintermapview.canvas_position_marker import CanvasPositionMarker

from src.algorithm.main_algorithm import Engine
from src.common import distance_operations
from src.gui import util
from src.model.node import Node

class MapTab(customtkinter.CTkFrame):
    # Algorithm arguments
    starting_index: int = 0
    destination_index: int = 0
    nodes: list[Node] = []  # List of available Nodes
    adj_matrix: list[list[float]] = []  # Two-dimensional list to hold the
adjacency matrix

    # Output
    route: list[Node] = []  # List of route Nodes
    distance: float = 0

    # Map attributes
    markers: list[CanvasPositionMarker] = []  # Markers represent nodes
    paths: list[CanvasPath] = []  # Paths represent edges

    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)
        button_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='bold')
        entry_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')

        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=0)
        self.grid_columnconfigure(1, weight=1)

        self.map_input_frame = MapInputFrame(master=self, width=180)
        self.map_input_frame.grid(row=0, column=0, padx=(10, 5), pady=10,
sticky='nswe')

        # Add path
        self.add_path_button =
customtkinter.CTkButton(master=self.map_input_frame,
                                                       text='Add Path',
                                                       font=button_font,
                                                       command=self.add_path_even
t)
        self.add_path_button.grid(row=5, column=0, padx=20, pady=(10, 0))

        # Clear map
        self.clear_map_button =
customtkinter.CTkButton(master=self.map_input_frame,
                                                        text='Clear Map',
                                                        font=button_font,
                                                        command=self.clear_map_ev
ent)
        self.clear_map_button.grid(row=6, column=0, padx=20, pady=(10, 0))

        # Start
        self.start_button = customtkinter.CTkButton(master=self.map_input_frame,
                                                    text='Start',
                                                    font=button_font,
                                                    command=self.find_shortest_ro
ute_and_visualize_route)
        self.start_button.grid(row=14, column=0, padx=20, pady=(10, 0))

        self.map_output_frame = MapOutputFrame(master=self)
        self.map_output_frame.grid(row=0, column=1, padx=(5, 10), pady=10,
sticky='nswe')

        # Search map entry
        self.search_map_entry =
customtkinter.CTkEntry(master=self.map_output_frame,
                                                       width=400,
                                                       placeholder_text='Search
map',
                                                       font=entry_font)
        self.search_map_entry.grid(row=0, column=0, padx=0, pady=10)
        self.search_map_entry.bind('<Return>', self.search_event)

        # Search map button
        self.search_map_button =
customtkinter.CTkButton(master=self.map_output_frame,
                                                         text='Search',
                                                         font=button_font,
                                                    command=self.search_even
t)
        self.search_map_button.grid(row=0, column=1, padx=0, pady=10)

        # Map widget
        self.map = tkintermapview.TkinterMapView(master=self.map_output_frame,
                                                 corner_radius=10)
        self.map.grid(row=1, rowspan=2, column=0, columnspan=3, padx=(0, 0),
pady=(0, 0), sticky='nswe')
        self.map.set_address('Lebak Siliwangi')
        self.map.add_right_click_menu_command(label="Add Marker",
                                              command=self.add_marker_event,
                                              pass_coords=True)

    def add_marker_event(self, coords) -> None:
        marker_id = len(MapTab.markers)
        marker_address = tkintermapview.convert_coordinates_to_address(coords[0],
coords[1])
        new_marker = self.map.set_marker(coords[0],
                                         coords[1],
                                         text=f'{marker_id}.
{marker_address.street}')

        MapTab.markers.append(new_marker)
        MapTab.nodes.append(Node(marker_id, coords[0], coords[1]))

    def add_path_event(self) -> None:
        # If there are less than two markers on the map
        if len(MapTab.markers) < 2:
            self.map_input_frame.map_message.configure(text='Not enough
markers.',
                                                       text_color='red')
            return

        # If the first node or the second node is not valid,
        # display error message until it is valid
        if not
(MapTab.is_index_valid(self.map_input_frame.first_node_entry.get()) and
                MapTab.is_index_valid(self.map_input_frame.second_node_entry.get(
))):

            # Clear status message
            self.map_input_frame.map_message.configure(text="")

            # If the first node is not valid
            if not
MapTab.is_index_valid(self.map_input_frame.first_node_entry.get()):
                self.map_input_frame.map_message.configure(text="Invalid first
node.",
                                                           text_color='red')
                return
            else:
                self.map_input_frame.map_message.configure(text="")

            # If the second node is not valid
            if not
MapTab.is_index_valid(self.map_input_frame.second_node_entry.get()):
```

```
                self.map_input_frame.map_message.configure(text="Invalid second
node.",
                                                           text_color='red')
                return
            else:
                self.map_input_frame.map_message.configure(text="")

        # If the first and second node is the same
        if int(self.map_input_frame.first_node_entry.get()) ==
int(self.map_input_frame.second_node_entry.get()):
            self.map_input_frame.map_message.configure(text='Cannot add path.',
                                                       text_color='red')
            return

        first_node_position: tuple[float, float] = 0.0, 0.0
        second_node_position: tuple[float, float] = 0.0, 0.0

        for node in MapTab.nodes:
            if node.node_id == int(self.map_input_frame.first_node_entry.get()):
                first_node_position = (node.x, node.y)
            if node.node_id == int(self.map_input_frame.second_node_entry.get()):
                second_node_position = (node.x, node.y)

        # If MapTab.paths is empty, add path without checking
        if len(MapTab.paths) == 0:
            new_path = self.map.set_path([first_node_position,
second_node_position])
            MapTab.paths.append(new_path)
            self.map_input_frame.map_message.configure(text="Path added.",
                                                       text_color='green')
            return

        # Check if the path already exists
        should_add = True
        for path in MapTab.paths:
            if (path.position_list[0] == first_node_position and
path.position_list[1] == second_node_position) or \
                    (path.position_list[0] == first_node_position and
path.position_list[1] == first_node_position):
                should_add = False

        if should_add:
            new_path = self.map.set_path([first_node_position,
second_node_position])
            MapTab.paths.append(new_path)
            self.map_input_frame.map_message.configure(text="Path added.",
                                                       text_color='green')
            return

        self.map_input_frame.map_message.configure(text="Path already exists.",
                                                   text_color='red')

    def clear_map_event(self) -> None:
        # Reset all class attribute
        MapTab.starting_index = 0
        MapTab.destination_index = 0
        MapTab.nodes = []
        MapTab.adj_matrix = []
        MapTab.route = []
        MapTab.distance = 0

        MapTab.markers = []
        MapTab.paths = []

        # Delete all markers and paths from the map
        self.map.delete_all_marker()
        self.map.delete_all_path()

        # Clear any output
        self.map_output_frame.result_tab_view.route_label.configure(text='')
        self.map_output_frame.result_tab_view.distance_label.configure(text='')
        self.map_input_frame.status_message.configure(text='')

        self.map_input_frame.map_message.configure(text='Map cleared.',
                                                   text_color='green')

    def search_event(self, event=None) -> None:
        self.map.set_address(self.search_map_entry.get())

    def find_shortest_route_and_visualize_route(self) -> None:
        # If there is less than 2 markers on the map
        if len(MapTab.markers) < 2:
            self.map_input_frame.status_message.configure(text="Not enough
markers.",
                                                          text_color='red')
            return

        # If there is no path
        if len(MapTab.paths) == 0:
            self.map_input_frame.status_message.configure(text="Add a path
first.",
                                                          text_color='red')
            return

        # If the starting node or the destination node is not valid,
        # display error message until it is valid
        if not
(MapTab.is_index_valid(self.map_input_frame.starting_node_entry.get()) and
                MapTab.is_index_valid(self.map_input_frame.destination_node_entry
.get())):

            # Clear status message
            self.map_input_frame.status_message.configure(text="")

            # If the starting node index is not valid
            if not
MapTab.is_index_valid(self.map_input_frame.starting_node_entry.get()):
                self.map_input_frame.status_message.configure(text="Invalid
starting node.",
                                                              text_color='red')
                return
            else:
                self.map_input_frame.status_message.configure(text="")

            # If the destination node index is not valid
            if not
MapTab.is_index_valid(self.map_input_frame.destination_node_entry.get()):
                self.map_input_frame.status_message.configure(text="Invalid
destination node.",
                                                              text_color='red')
                return
            else:
                self.map_input_frame.status_message.configure(text="")

        try:
            self.find_shortest_route()
        except TypeError:
            # There is no route from the starting node to the destination node
            self.map_input_frame.status_message.configure(text="No route found.",
                                                          text_color='red')
            return

        self.visualize_route()
        self.show_route_and_distance()

    def find_shortest_route(self) -> None:
        MapTab.starting_index =
int(self.map_input_frame.starting_node_entry.get())
        MapTab.destination_index =
int(self.map_input_frame.destination_node_entry.get())

        MapTab.fill_adj_matrix()

        # Start algorithm
        if self.map_input_frame.algorithm_options.get() == 'A*':
            # A-star path-finding
            MapTab.distance, MapTab.route =
Engine.search_astar(MapTab.starting_index,
                                                                MapTab.destinatio
n_index,
                                                                MapTab.nodes,
                                                                MapTab.adj_matrix
)
        else:
            # UCS path-finding
            MapTab.distance, MapTab.route =
Engine.search_ucs(MapTab.starting_index,
                                                              MapTab.destination_
index,
                                                              MapTab.nodes,
                                                              MapTab.adj_matrix)

    @staticmethod
    def fill_adj_matrix() -> None:
        # First, we clear the adjacency matrix
        MapTab.adj_matrix = []

        # Then, we create a list of tuples.
        # Each tuple represents the nodes that are adjacent
        # The integer values in the tuples are node IDs
        adjacent_nodes: list[tuple[Node, Node]] = []
        while len(adjacent_nodes) < len(MapTab.paths):
            for path in MapTab.paths:
                first_node = None
                second_node = None

                for node in MapTab.nodes:
                    if path.position_list[0] == (node.x, node.y):
                        first_node = node
                        break

                for node in MapTab.nodes:
                    if path.position_list[1] == (node.x, node.y):
                        second_node = node
                        break

                adjacent_nodes.append((first_node, second_node))

        # Fill adj_matrix with 0.0
        for i in range(len(MapTab.nodes)):
            MapTab.adj_matrix.append([0.0 for _ in range(len(MapTab.nodes))])

        # Fill adj_matrix with the geodesic distance as weights
        for adj_node in adjacent_nodes:
            MapTab.adj_matrix[adj_node[0].node_id][adj_node[1].node_id] = \
                distance_operations.geodesic_distance((adj_node[0].x,
adj_node[0].y), (adj_node[1].x, adj_node[1].y))

            MapTab.adj_matrix[adj_node[1].node_id][adj_node[0].node_id] = \
                distance_operations.geodesic_distance((adj_node[0].x,
adj_node[0].y), (adj_node[1].x, adj_node[1].y))

    def visualize_route(self) -> None:
        # Resets all the path color
        new_map_paths = []
        for path in MapTab.paths:
            new_map_paths.append(self.map.set_path(position_list=path.position_li
st))
        MapTab.paths = new_map_paths

        # Color the route paths
        i = 0
        while i < len(MapTab.route) - 1:
            for path in MapTab.paths:
                if i + 1 < len(MapTab.route):
                    if (path.position_list[0] == (MapTab.route[i].x,
MapTab.route[i].y) and
                        path.position_list[1] == (MapTab.route[i + 1].x,
MapTab.route[i + 1].y)) or \
                            (path.position_list[1] == (MapTab.route[i].x,
MapTab.route[i].y) and
                             path.position_list[0] == (MapTab.route[i + 1].x,
MapTab.route[i + 1].y)):
                        MapTab.paths.remove(path)
                        MapTab.paths.append(self.map.set_path(position_list=path.
position_list,
                                                              color='yellow
green'))
                        i += 1
                else:
                    break

        self.map_input_frame.status_message.configure(text='Route visualized.',
                                                      text_color='green')

    def show_route_and_distance(self) -> None:
        # Convert the route to string, example: 2-1-6-3-7
        route = ''
        for route_node in MapTab.route:
            route += f'{route_node.node_id}-'
        route = route[:-1]  # Remove trailing hyphen

        self.map_output_frame.result_tab_view.route_label.configure(text=route)
        self.map_output_frame.result_tab_view.distance_label.configure(text=f'{Ma
pTab.distance:.5f} m')

    @staticmethod
    def is_index_valid(index: int) -> bool:
        """
        Validates the node index
        :param index: marker's index/label
        :return: True if the index is valid, False otherwise
        """
        try:
            if 0 <= int(index) <= len(MapTab.markers) - 1:
                return True
        except ValueError:
            return False

        return False

class MapInputFrame(customtkinter.CTkFrame):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Custom-defined fonts
        title_font = customtkinter.CTkFont(family='Segoe UI', size=-18,
weight='bold')
        message_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
        input_label_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='bold')
        entry_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
        select_theme_font = customtkinter.CTkFont(family='Segoe UI', size=-13,
weight='normal')
        select_algorithm_font = customtkinter.CTkFont(family='Segoe UI', size=-
13, weight='normal')

        # Expandable empty space between the Start button and UI theme options
        self.grid_rowconfigure(16, weight=1)

        # App title
        self.app_title = customtkinter.CTkLabel(master=self,
                                                text='Compass',
                                                font=title_font)
        self.app_title.grid(row=0, column=0, padx=20, pady=(20, 10))
        # First node
        self.first_node = customtkinter.CTkLabel(master=self,
                                                 text='First node:',
                                                 font=input_label_font)
        self.first_node.grid(row=1, column=0, padx=20, pady=0)

        self.first_node_entry = customtkinter.CTkEntry(master=self,
                                                       placeholder_text="Marker
number",
                                                       font=entry_font)
        self.first_node_entry.grid(row=2, column=0, padx=20, pady=0)

        # Second node
        self.second_node_label = customtkinter.CTkLabel(master=self,
                                                        text='Second node:',
                                                        font=input_label_font)
        self.second_node_label.grid(row=3, column=0, padx=20, pady=0)

        self.second_node_entry = customtkinter.CTkEntry(master=self,
                                                        placeholder_text="Marker
number",
                                                        font=entry_font)
        self.second_node_entry.grid(row=4, column=0, padx=20, pady=(0, 5))

        # Add path button is defined in MapTab

        # Clear map button is defined in MapTab

        self.map_message = customtkinter.CTkLabel(master=self,
                                                  text='',
                                                  font=message_font)
        self.map_message.grid(row=7, column=0, padx=20, pady=(0, 0))

        # Starting node and destination node
        self.starting_node_label = customtkinter.CTkLabel(master=self,
                                                          text='Starting node:',
                                                          font=input_label_font)
        self.starting_node_label.grid(row=8, column=0, padx=20, pady=0)

        self.starting_node_entry = customtkinter.CTkEntry(master=self,
                                                          placeholder_text="Marke
r number",
                                                          font=entry_font)
        self.starting_node_entry.grid(row=9, column=0, padx=20, pady=0)

        self.destination_node_label = customtkinter.CTkLabel(master=self,
                                                             text='Destination
node:',
                                                             font=input_label_fon
t)
        self.destination_node_label.grid(row=10, column=0, padx=20, pady=0)

        self.destination_node_entry = customtkinter.CTkEntry(master=self,
                                                             placeholder_text="Ma
rker number",
                                                             font=entry_font)
        self.destination_node_entry.grid(row=11, column=0, padx=20, pady=(0, 5))
        # Select algorithm
        self.select_algorithm_label = customtkinter.CTkLabel(master=self,
                                                             text="Select
algorithm:",
                                                             font=input_label_fon
t)
        self.select_algorithm_label.grid(row=12, column=0, padx=20, pady=(5, 0))

        self.algorithm_options = customtkinter.CTkOptionMenu(master=self,
                                                             values=['A*',
'UCS'],
                                                             font=select_algorith
m_font)
        self.algorithm_options.grid(row=13, column=0, padx=20, pady=(0, 10))

        # Start button is defined in MapTab

        self.status_message = customtkinter.CTkLabel(master=self,
                                                     text='',
                                                     font=message_font)
        self.status_message.grid(row=15, column=0, padx=20, pady=0)

        # Select GUI theme
        self.select_theme_label = customtkinter.CTkLabel(master=self,
                                                         text="Select theme:",
                                                         font=select_theme_font)
        self.select_theme_label.grid(row=17, column=0, padx=20, pady=0)

        self.theme_options = customtkinter.CTkOptionMenu(master=self,
                                                         values=["Dark", "Light",
"System"],
                                                         font=select_theme_font,
                                                         command=util.change_appe
arance_mode)
        self.theme_options.grid(row=18, column=0, padx=20, pady=(0, 20))

class MapOutputFrame(customtkinter.CTkFrame):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Configure the grid system
        self.grid_rowconfigure(2, weight=1)
        self.grid_columnconfigure((0, 2), weight=1)
        self.grid_rowconfigure(1, weight=0)
        self.grid_columnconfigure(1, weight=0)
        self.grid_rowconfigure(3, weight=0)

        # Search map entry is defined in MapTab

        # Search map button is defined in MapTab

        # Map widget is defined in MapTab

        # Result tab view
        self.result_tab_view = MapResultTabView(master=self,
                                                height=125)
        self.result_tab_view.grid(row=3, column=0, columnspan=3, padx=20,
pady=20, sticky='we')
```

```
        self.result_tab_view._segmented_button.configure(font=('Segoe UI', -13,
'bold'))

class MapResultTabView(customtkinter.CTkTabview):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        output_font = customtkinter.CTkFont(family='Segoe UI', size=-14,
weight='normal')

        # Create two tabs for output
        self.add('Route')
        self.add('Distance')

        self.tab('Route').grid_columnconfigure(0, weight=1)
        self.tab('Distance').grid_columnconfigure(0, weight=1)

        # Create a scrollable frame to hold the route
        self.route_scrollable_frame =
customtkinter.CTkScrollableFrame(master=self.tab('Route'),
                                                                     height=30,
                                                                     orientatio
n='horizontal')
        self.route_scrollable_frame.grid(row=0, column=0, columnspan=1, pady=10,
sticky='ns')

        self.route_label =
customtkinter.CTkLabel(master=self.route_scrollable_frame,
                                                  text='',
                                                  text_color='green',
                                                  font=output_font)
        self.route_label.grid(row=0, column=0, padx=10)

        # Create a label for distance
        self.distance_label = customtkinter.CTkLabel(master=self.tab('Distance'),
                                                     text='',
                                                     text_color='green',
                                                     font=output_font)
        self.distance_label.grid(row=0, column=0, pady=(20, 0))
```

2.6 util.py

```
import customtkinter

def change_appearance_mode(new_appearance_mode: str) -> None:
    """
    Changes the GUI theme
    :param new_appearance_mode: string: "dark", "light", or "system"
    """
    customtkinter.set_appearance_mode(new_appearance_mode)
```

2.7 file_handler.py

```
from src.model.node import Node

class FileInputHandler:
    @staticmethod
    def load_file(path: str) -> (list[Node], list[list[float]]):
        file1 = open(path, 'r')

        lines = file1.readlines()
        node_count = -1
        node_list: list[Node] = []
        adj_matrix: list[list[float]] = []

        for i, line in enumerate(lines):
            try:
                if i == 0:  # matrix size / node count
                    node_count = int(line)
                    adj_matrix = [[] for _ in range(node_count)]
                    continue

                if 0 < i < node_count + 1:
                    node_id = i - 1
                    coordinates = line.split(' ')
                    if len(coordinates) != 2:
                        raise RuntimeError("Error while processing file.")
                    node_list.append(Node(node_id, float(coordinates[0]),
float(coordinates[1])))

                    continue

                if i >= node_count + 1:
                    weights = line.split(' ')
                    if len(weights) != node_count:
                        raise RuntimeError("Error while processing file.")

                    v_idx = i - (node_count + 1)
                    adj_matrix[v_idx] = ([0.0 for _ in range(node_count)])
                    for j, weight in enumerate(weights):
                        adj_matrix[v_idx][j] = float(weight)

            except (ValueError, IndexError):
                raise RuntimeError("Error while processing file.")

        return node_list, adj_matrix
```

2.8 node.py

```
from __future__ import annotations

class Node:
    def __init__(self, node_id: int, x: float, y: float):
        # a node must be retrievable from node list by its node id
        self.node_id = node_id
        self.x = x
        self.y = y

    def __eq__(self, compare_object: Node) -> bool:
        return self.node_id == compare_object.node_id

    def __str__(self):
        return str(self.node_id)
```

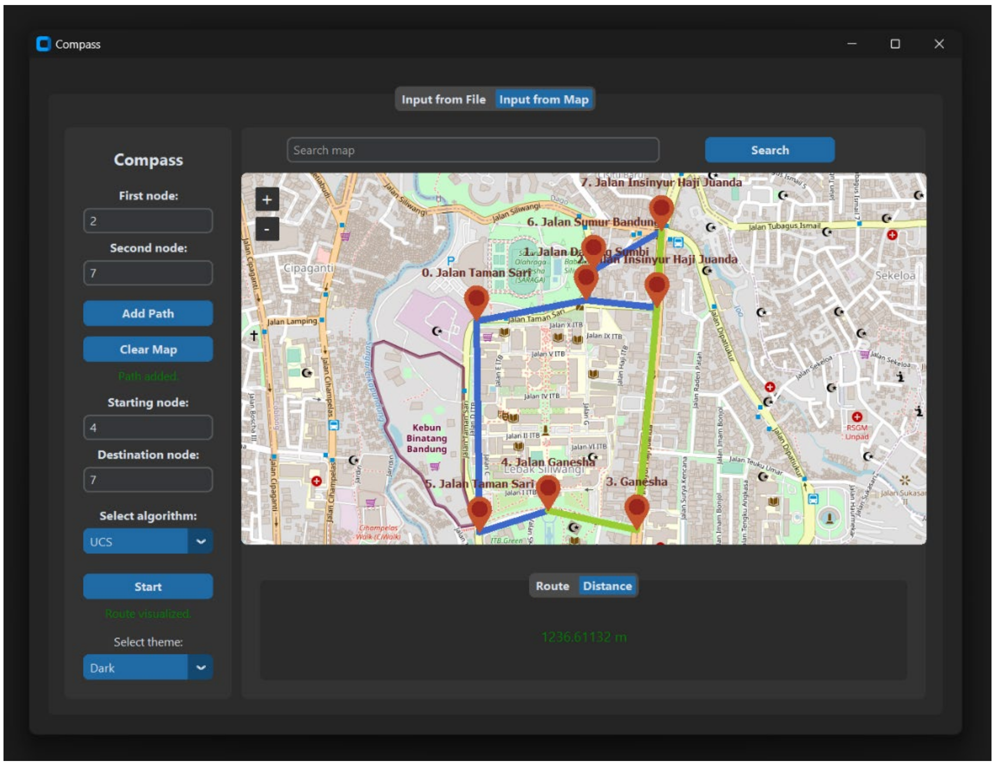
2.9 compass.py

```python
from src.gui import main_window

if __name__ == '__main__':
    app = main_window.MainWindow()
    app.mainloop()
```

BAB III
PENGUJIAN PROGRAM

3.1 Pengujian Lintasan Terpendek di Daerah ITB Kampus Ganesha



Compass

First node: 2

Second node: 7

Path added.

Starting node: 4

Destination node: 7

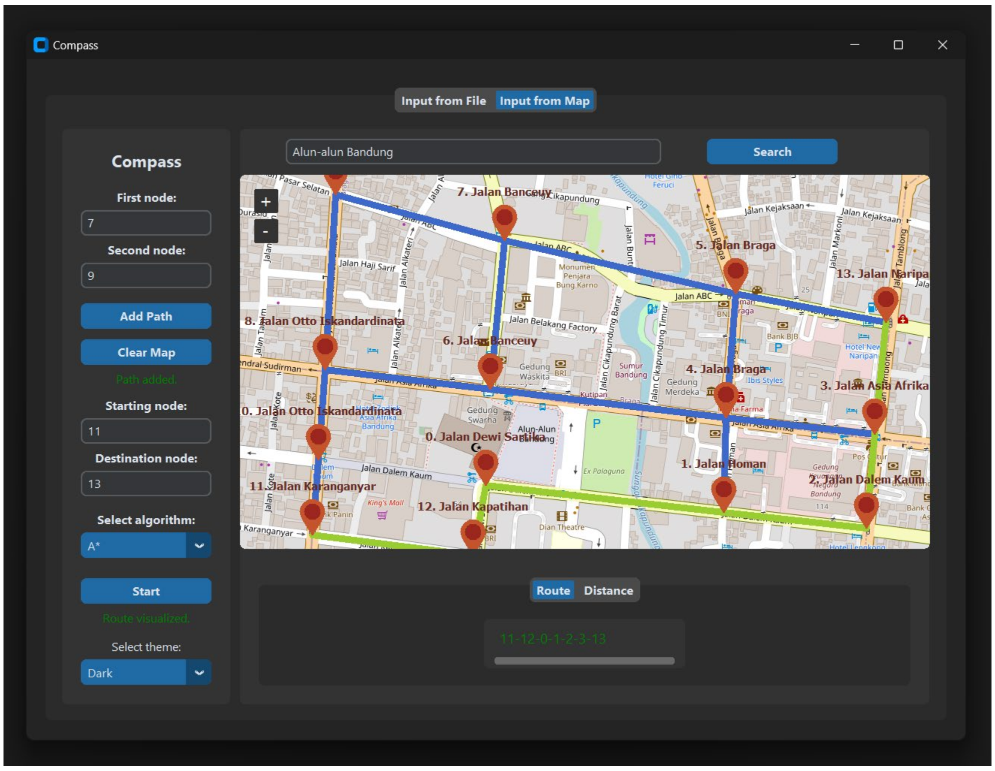
Select algorithm: UCS

Route visualized.

Select theme: Dark

1236.61132 m

3.2 Pengujian Lintasan Terpendek di Daerah Alun-alun Bandung



Alun-alun Bandung

Compass

First node: 7

Second node: 9

Path added.

Starting node: 11

Destination node: 13

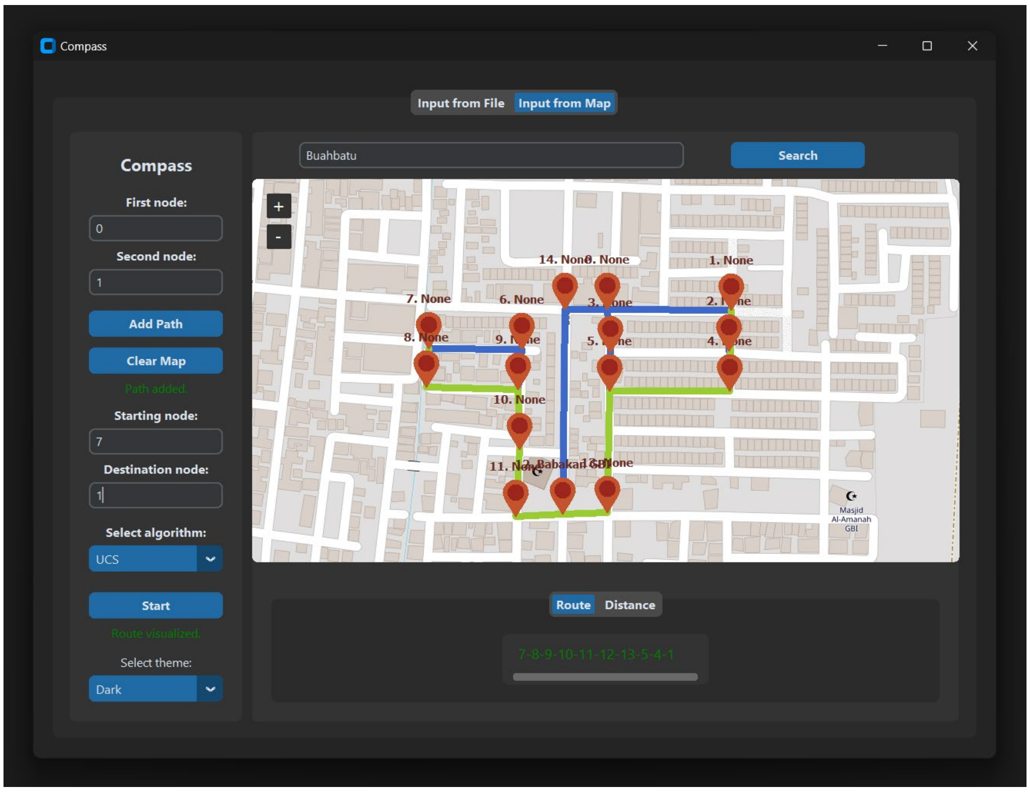
Select algorithm: A*

Route visualized.

Select theme: Dark

11-12-0-1-2-3-13

3.3 Pengujian Lintasan Terpendek di Daerah Buahbatu



Compass

Input from File | Input from Map

Search: Buahbatu

First node: 0

Second node: 1

Path added.

Starting node: 7

Destination node: 1

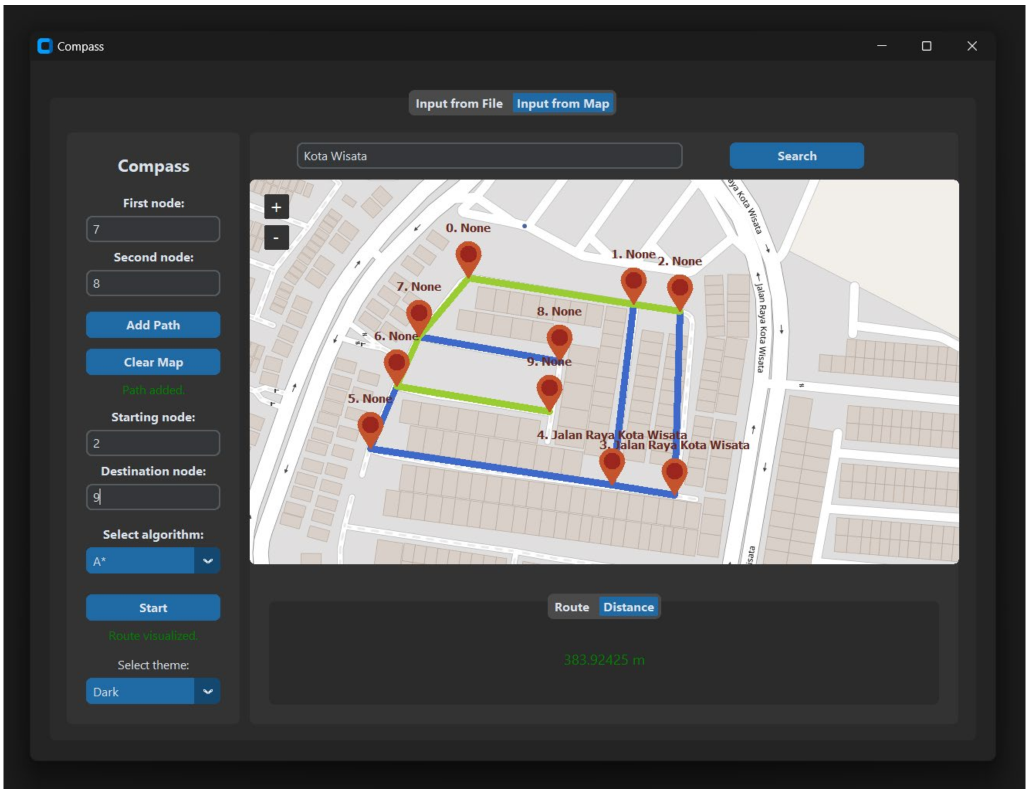
Select algorithm: UCS

Route visualized.

Select theme: Dark

Route: 7-8-9-10-11-12-13-5-4-1

3.4 Pengujian Lintasan Terpendek di Daerah Perumahan Kota Wisata



Compass

Input from File | Input from Map

Search: Kota Wisata

First node: 7

Second node: 8

Path added.

Starting node: 2

Destination node: 9

Select algorithm: A*

Route visualized.

Select theme: Dark

Distance: 383.92425 m

BAB IV
PENUTUP

4.1 Kesimpulan

Kami berhasil berhasil mengimplementasikan algoritma UCS dan A* untuk menyelesaikan masalah dalam menemukan lintasan terpendek dari suatu titik ke titik lain.

4.2 Komentar

Mengingat kondisi mahasiswa-mahasiswa Prodi Teknik Informatika angkatan 2021 yang sedang dilanda tugas, kami menyarankan beban tugas kecil untuk diperingan. Selain itu, kami ingin mengkritik spesifikasi tugas yang terlalu *vague* dan *loose*. Ini menyebabkan banyak kebingungan dan inkonsistensi di antara mahasiswa. Selain itu, nampaknya *sheets* QnA tidak dimonitor secara konsisten oleh asisten-asisten sehingga banyak pertanyaan-pertanyaan yang tidak terjawab.

LAMPIRAN

Remote repository

Berikut adalah tautan *remote repository* yang berisi *source code* untuk tugas ini.

https://github.com/noelsimbolon/Tucil3_13521096_13521163

***Checklist* program**

Poin	Ya	Tidak
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	✓	