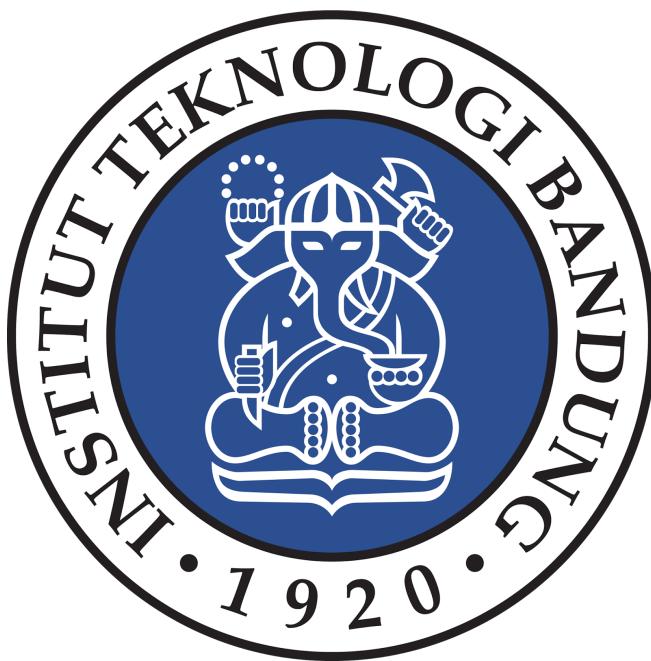


**LAPORAN
TUGAS KECIL 3 IF2211
STRATEGI ALGORITMA**

**Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan
Terpendek**



oleh

**Salomo Reinhart Gregory Manalu 13521063
Austin Gabriel Pardosi 13521084**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2022 / 2023**

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
PENJELASAN ALGORITMA PROGRAM	3
1.1 Deskripsi Permasalahan Rute Terpendek dan Pencarian Solusi	3
1.2 Implementasi Algoritma UCS dan A* untuk Mencari Solusi	3
1.2.1 Tahap Pra-proses Pencarian Solusi	3
1.2.2 Tahap Pencarian Solusi	4
1.3 Analisis Kompleksitas Waktu Algoritma UCS dan A*	4
1.4 Implementasi Bonus pada Program	5
1.4.1 Penggunaan TkinterMapView	5
1.4.2 Penggambaran Simpul pada Map	6
BAB II	7
SOURCE CODE PROGRAM	7
2.1 File algorithm.py	7
2.1 File parse_into_graph.py	8
2.3 File main.py	11
BAB III	16
EKSPERIMEN DAN TESTING	16
3.1 Testing untuk Peta daerah Alun-Alun Bandung	16
3.2 Testing untuk Peta daerah Buah Batu	17
3.3 Testing untuk Peta daerah ITB	18
3.4 Testing untuk Peta daerah Ring Road Medan	19
DAFTAR PUSTAKA	20
LAMPIRAN	21
<i>Link Repository Program</i>	21
<i>Checklist Tugas Kecil</i>	21

BAB I

PENJELASAN ALGORITMA PROGRAM

1.1 Deskripsi Permasalahan Rute Terpendek dan Pencarian Solusi

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Diberikan sebuah peta Google Map daerah yang ada di Bandung, lalu akan ditentukan lintasan terpendek berdasarkan peta Google Map . 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.

1.2 Implementasi Algoritma *UCS* dan *A** untuk Mencari Solusi

1.2.1 Tahap Pra-proses Pencarian Solusi

Diasumsikan bahwa program telah menerima sebuah file .txt masukan yang berisi jumlah simpul, koordinat setiap simpul, dan matriks ketetanggaan (bukan weighted). Data tersebut merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan data yang yang, 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.

1.2.2 Tahap Pencarian Solusi

Setelah menerima file input .txt yang berisi jumlah simpul, koordinat setiap simpul, dan matriks ketetanggaan (bukan weighted), program akan melakukan parsing terhadap file .txt tersebut. Program akan menerima N buah simpul beserta koordinat dari masing-masing simpul. Program juga akan menerima matriks ketetanggaan (bukan weighted adjacency matrix). Parsing pertama adalah mengubah matriks ketetanggaan tersebut menjadi weighted adjacency matrix. Setiap simpul juga akan menyimpan koordinatnya (hasil parsing pertama). Setelah menerima adjacency matrix, akan dilakukan proses parsing kedua, yaitu mengubah weighted adjacency matrix menjadi sebuah graf. Setelah graf selesai di parsing, program siap untuk menerima input dari user. Program akan menerima 2 input, yaitu start node dan goal node. Setelah menerima start dan goal node, user akan memilih algoritma mana yang akan dipakai untuk mencari rute terpendek (UCS atau A*). Misalkan user memilih algoritma A*, maka program akan menjalankan algoritma A* dan mencari rute terpendek antara start node dan goal node. Program akan mengembalikan output berupa path terpendek dan cost dari path tersebut. Hal yang sama juga akan terjadi apabila user memilih algoritma UCS.

Pada algoritma A*, digunakan sebuah fungsi heuristik, yaitu Euclidean distance. Euclidean distance akan menghitung jarak node yang sedang dikunjungi ke goal node (jarak ditarik garis lurus). Sedangkan pada algoritma UCS, tidak digunakan fungsi heuristik apapun.

1.3 Analisis Kompleksitas Waktu Algoritma *UCS* dan *A**

Algoritma Uniform Cost Search (UCS) dan A* adalah algoritma untuk mencari jalur atau path terpendek dari start node hingga ke goal node pada sebuah graf berbobot. Kompleksitas waktu algoritma UCS adalah $O(bm)$ untuk kasus terbaik. b adalah faktor percabangan

maksimum dan m adalah jumlah maksimum sisi pada graf. Untuk kasus terburuk, kompleksitas waktu algoritma UCS adalah $O(b^d)$, dimana d adalah jarak start node hingga ke goal node (kedalaman).

Berbeda dengan algoritma UCS, algoritma A* memiliki fungsi heuristik yang dapat membantu algoritma untuk dapat menemukan jalur terpendek dengan waktu yang lebih cepat. Kompleksitas waktu terbaik algoritma A* adalah $O(1)$. Hal ini dapat terjadi apabila fungsi heuristik sangat akurat dan tujuan ditemukan di simpul awal. Pada kasus terburuk, kompleksitas waktu terbaik algoritma A* adalah $O(b^d)$, dimana b adalah faktor percabangan maksimum dan m adalah jumlah maksimum sisi pada graf dan d adalah kedalaman atau jarak terpendek dari start node hingga ke goal node.

Terkait kecepatan pencarian solusi oleh algoritma UCS dan algoritma A* berbeda dan bergantung pada test case yang diujikan. A* biasanya lebih cepat daripada UCS pada kasus graph yang besar dikarenakan fungsi heuristik yang dipakai pada A* sehingga mempercepat pencarian solusi optimal. Namun pada kasus graph yang kecil, UCS dapat bekerja lebih cepat daripada A*. Kecepatan pencarian solusi algoritma UCS dan A* dipengaruhi oleh banyak faktor seperti ukuran graf, kepadatan graf, serta jumlah simpul dan sisi.

1.4 Implementasi Bonus pada Program

1.4.1 Penggunaan TkinterMapView

TkinterMapView adalah salah satu library Python yang memungkinkan untuk menggambar peta dan tampilan dunia nyata dengan menggunakan Tkinter sebagai antarmuka pengguna. Kami menggunakan TkinterMapView dikarenakan penggunaannya yang tidak

memerlukan koneksi internet yang membantu penampilan peta secara offline, biayanya yang gratis, serta kemudahannya untuk diintegrasikan sehingga GUI dapat menampilkan peta dengan mudah menggunakan tampilan OpenStreetMap.

1.4.2 Penggambaran Simpul pada Map

Penggambaran simpul pada map menggunakan library TkinterMapView. Implementasi penggambaran simpul ini dimulai dengan menggambarkan seluruh simpul yang ada pada konfigurasi file txt berdasarkan hasil dari file parser ke dalam Map berdasarkan koordinat (lintang dan bujur) masing-masing simpul. Langkah selanjutnya adalah mengambil nilai yang dihasilkan oleh algoritma UCS ataupun A*. Terakhir kami menampilkan sisi terpendek dan meletakkannya pada peta dan memberi warna biru sebagai penanda.

BAB II

SOURCE CODE PROGRAM

2.1 File *algorithm.py*

```
from heapq import heappop, heappush
import math
import parse_into_graph as parser

class UCS :
    def ucs(start, goal, graph):
        nodesToExplore = [(0, start)]
        visited = []
        cost = {start: 0}
        parents = {start: None}

        while nodesToExplore:
            currentCost, currentNode = heappop(nodesToExplore)
            if currentNode == goal:
                path = []
                while currentNode:
                    path.append(currentNode)
                    currentNode = parents[currentNode]
                return path[::-1], cost[goal]

            visited.append(currentNode)

            for neighbor in graph.neighbors(currentNode):
                if neighbor in visited:
                    continue

                newCost = cost[currentNode] + graph[currentNode][neighbor]['weight']
                if neighbor not in cost or newCost < cost[neighbor]:
                    cost[neighbor] = newCost
                    parents[neighbor] = currentNode
                    heappush(nodesToExplore, (newCost, neighbor))

        return [], 0
```

Gambar 2.1.1 Source Code File *algorithm.py*

```

class aStar :
    def find_euclidean_distance(a, b):
        x1, y1 = a
        x2, y2 = b
        return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

    def astar(start, goal, graph, nodes, heuristic_func):
        exploreNodes = [(0, start)]
        parents = {start: None}
        currentPathWithCost = {start: 0}

        while exploreNodes:
            currentCost, currentNode = heappop(exploreNodes)

            if currentNode == goal:
                shortestPath = []
                gn = currentPathWithCost[currentNode]
                while currentNode:
                    shortestPath.append(currentNode)
                    currentNode = parents[currentNode]
                return shortestPath[::-1], gn

            for neighbor in graph.neighbors(currentNode):
                gn = currentPathWithCost[currentNode] + graph[currentNode][neighbor]['weight']

                if heuristic_func:
                    hn = aStar.find_euclidean_distance(nodes[neighbor], nodes[goal])
                    fn = gn + hn

                if (neighbor not in currentPathWithCost) or (gn < currentPathWithCost[neighbor]):
                    currentPathWithCost[neighbor] = gn
                    parents[neighbor] = currentNode
                    heappush(exploreNodes, (fn, neighbor))

        return [], 0

```

Gambar 2.1.2 Source Code File algorithm.py

2.1 File parse_into_graph.py

```

import math
import networkx as nx

def distance(p1, p2):
    lat1, lon1 = p1
    lat2, lon2 = p2
    return math.sqrt((lat1-lat2)**2 + (lon1-lon2)**2)

```

Gambar 2.2.1 Source Code File parse_into_graph.py

```

def parse_into_adjacency_mtr(filename):
    # Parse nodes and coordinates
    nodes = {}
    with open(filename) as f:
        lines = f.readlines()
    try :
        n = int(lines[0].strip())
        listnodes = []
        for line in lines[1:n+1]:
            label, lat, lon = line.strip().split()
            nodes[label] = (float(lat), float(lon))
            listnodes.append(label)

        if (len(listnodes) < 8) :
            raise ValueError("There should be at least 8 nodes.")
        if (len(listnodes) != n) :
            raise ValueError(f'There should be {n} nodes')

        m = n + 1

        matrix = lines[m:]
        mtr = [[int(x) for x in line.split()] for line in matrix]

        if len(mtr) != n or len(mtr[0]) != n:
            raise ValueError("The adjacency matrix should be a square matrix with dimensions n x n.")
        if any(mtr[i][i] != 0 for i in range(n)):
            raise ValueError("The diagonal elements of the adjacency matrix should be zero.")

        for i in range(n) :
            for j in range(n) :
                if mtr[i][j] == 1 :
                    p1 = nodes[listnodes[j]]
                    p2 = nodes[listnodes[i]]
                    (variable) mtr: list[list[int]]
```

return mtr, nodes, listnodes

```

except FileNotFoundError:
    print(f"Error: file '{filename}' not found.")
except ValueError as e:
    print(f"Error: {e}")

```

Gambar 2.2.2 Source Code File `parse_into_graph.py`

```

def parse_adjacency_matrix(adj_matrix, listnodes):
    graph = nx.Graph()
    count_nodes = len(adj_matrix)

    # Add nodes to graph
    for i in range(count_nodes):
        node = listnodes[i]
        graph.add_node(node)

    # Add edges to graph
    for i in range(count_nodes):
        for j in range(count_nodes):
            if adj_matrix[i][j] != 0:
                node1 = listnodes[i]
                node2 = listnodes[j]
                weight = adj_matrix[i][j]
                graph.add_edge(node1, node2, weight=weight)

    return graph

def print_graph(graph):
    for node in graph.nodes():
        neighbors = list(graph.neighbors(node))
        if len(neighbors) > 0:
            print("{} -> {}".format(node, end=""))
            for neighbor in neighbors[:-1]:
                weight = graph.get_edge_data(node, neighbor)['weight']
                print("{} ({})".format(neighbor, weight), end=", ")
            last_neighbor = neighbors[-1]
            weight = graph.get_edge_data(node, last_neighbor)['weight']
            print("{} ({})".format(last_neighbor, weight))
        else:
            print(node)

```

Gambar 2.2.3 Source Code File parse_into_graph.py

2.3 File main.py

```
from tkinter import filedialog, ttk
from PIL import Image
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkintermapview import TkinterMapView
from algorithm import UCS, aStar
import matplotlib.pyplot as plt
import customtkinter
import tkinter
import tkinter as tk
import os
import time
import networkx as nx
import parse_into_graph as p

# Digunakan untuk meset tema warna GUI
customtkinter.set_default_color_theme("blue")

# Digunakan untuk membuka path gambar untuk logo pada button
PATH = os.path.dirname(os.path.realpath(__file__))
```

Gambar 2.3.1 Source Code File main.py

```
class PathFinder(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        # Membuat window
        self.title("Path Map Finder")
        self.geometry(f"{1100}x{580}")

        # Konfigurasi grid layout (1x2)
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)

        # Konfigurasi image
        self.add_folder_image = self.load_image("..\img\add_folder.png", 30)
        self.execute_image = self.load_image("..\img\execute.png", 30)

        # Membuat sidebar frame with widgets
        self.sidebar_frame = customtkinter.CTkFrame(self, corner_radius=10, width=200)
        self.sidebar_frame.grid(row=0, column=0, sticky="nsew", padx=(20,0), pady=20)
        self.sidebar_frame.grid_rowconfigure(15, weight=1) #frame maksimal
        self.sidebar_frame.grid_rowconfigure(12, weight=20)

        # Membuat Main Label
        self.logo_label = customtkinter.CTkLabel(self.sidebar_frame, text="Path Map Finder", font=customtkinter.CTkFont(size=20, weight="bold"))
        self.logo_label.grid(row=0, column=0, padx=10, pady=(20,20))

        # Membuat Insert File Button
        self.file_is_selected = False
        self.Map_label = customtkinter.CTkLabel(self.sidebar_frame, text="Select Map: ", font=customtkinter.CTkFont(size=20, weight="bold"))
        self.Map_label.grid(row=1, column=0, padx=25, pady=(20,0), sticky="w")
        self.sidebar_button_1 = customtkinter.CTkButton(self.sidebar_frame, height=40, width=130, image=self.add_folder_image, text="Insert File", command=self.select_map)
        self.sidebar_button_1.grid(row=2, column=0, padx=10, pady=(10,5))
        self.file_info = customtkinter.CTkLabel(self.sidebar_frame, text="")
        self.file_info.grid(row=3, column=0, padx=0, pady=0, sticky="n")

        # Membuat combobox
        self.Node_label = customtkinter.CTkLabel(self.sidebar_frame, text="Nodes Search: ", font=customtkinter.CTkFont(size=20, weight="bold"))
        self.Node_label.grid(row=4, column=0, padx=25, pady=(30,0), sticky="w")
        self.combobox_1 = customtkinter.CTkComboBox(values=[""], master= self.sidebar_frame)
        self.combobox_1.grid(row=5, column=0, padx=0, pady=10)
        self.combobox_2 = customtkinter.CTkComboBox(values=[""], master= self.sidebar_frame,)
        self.combobox_2.grid(row=6, column=0, padx=0, pady=10)
```

Gambar 2.3.2 Source Code File main.py

```

# Membuat Radiobutton
self.radio_var = tkinter.IntVar(value=0)
self.label_radio_group = customtkinter.CTkLabel(self.sidebar_frame, text="Algorithm:", font=customtkinter.CTkFont(size=20, weight="bold"))
self.label_radio_group.grid(row=7, column=0, padx=30, pady=(40,0), sticky="w")
self.radio_button_1 = customtkinter.CTkRadioButton(self.sidebar_frame, variable=self.radio_var, value=0, text="A*")
self.radio_button_1.grid(row=8, column=0, padx=40, pady=10, sticky="nw")
self.radio_button_2 = customtkinter.CTkRadioButton(self.sidebar_frame, variable=self.radio_var, value=1, text="UCS")
self.radio_button_2.grid(row=9, column=0, padx=40, pady=10, sticky="nw")

# Membuat Button
self.sidebar_button_2 = customtkinter.CTkButton(self.sidebar_frame, height=40, width=130, text="Execute", image=self.execute_image, command=self.execute)
self.sidebar_button_2.grid(row=10, column=0, padx=10, pady=(40,5))
self.error_info = customtkinter.CTkLabel(self.sidebar_frame, text="")
self.error_info.grid(row=11, column=0, padx=0, pady=0, sticky="n")

self.appearance_mode_label = customtkinter.CTkLabel(self.sidebar_frame, text="Theme Mode:", anchor="w", font=customtkinter.CTkFont(size=15, weight="bold"))
self.appearance_mode_label.grid(row=13, column=0, padx=0, pady=(10,0), sticky="s")
self.appearance_mode_optionMenu = customtkinter.CTkOptionMenu(self.sidebar_frame, values=["Light", "Dark"], command=self.change_appearance_mode_event)
self.appearance_mode_optionMenu.grid(row=14, column=0, padx=10, pady=(10,10))

# Membuat main frame
self.main_frame = customtkinter.CTkFrame(self, corner_radius=10)
self.main_frame.grid(row=0, column=1, sticky="nsew", padx=20, pady=20)

# Membuat tabview
self.tabview = customtkinter.CTkTabview(self.main_frame, width=1230, height=700)
self.tabview.grid(row=0, column=0, padx=(20, 0), pady=(20, 0), sticky="nsew")
self.tabview.add("Graph")
self.tabview.add("Map")
self.tabview.tab("Graph").grid_columnconfigure(3, weight=1) # configure grid of individual tabs
self.tabview.tab("Graph").grid_columnconfigure(1, weight=2) # configure grid of individual tabs
self.tabview.tab("Graph").grid_rowconfigure(8, weight=5) # configure grid of individual tabs
self.tabview.tab("Map").grid_rowconfigure(1, weight=1)
self.tabview.tab("Map").grid_rowconfigure(0, weight=0)
self.tabview.tab("Map").grid_columnconfigure(0, weight=1)
self.tabview.tab("Map").grid_columnconfigure(1, weight=0)
self.tabview.tab("Map").grid_columnconfigure(2, weight=1)

# Membuat Label Jawaban
self.algorithm_label = customtkinter.CTkLabel(self.tabview.tab("Graph"), text="")
self.algorithm_label.grid(row=0, column=0, padx=50, pady=(20,20))

# Membuat frame untuk graph
self.graph_frame = customtkinter.CTkFrame(self.tabview.tab("Graph"), corner_radius=10)
self.graph_frame.grid(row=1, column=0, padx=50, pady=10, rowspan=7, sticky="n")

```

Gambar 2.3.3 Source Code File main.py

```

# Membuat lokasi peletakan graph
self.place = customtkinter.CTkFrame(self.graph_frame, corner_radius=10)
self.place.pack(expand=True, fill=tk.BOTH, anchor=tk.CENTER)

# Membuat Graph Info Label
self.result_label = customtkinter.CTkLabel(self.tabview.tab("Graph"), text="", font=customtkinter.CTkFont(size=20, weight="bold"))
self.result_label.grid(row=0, column=2)
self.graph_path_label = customtkinter.CTkLabel(self.tabview.tab("Graph"), text="", font=customtkinter.CTkFont(size=20, weight="bold"))
self.graph_path_label.grid(row=1, column=2, sticky="nw", pady=(0,20))
self.cost_label = customtkinter.CTkLabel(self.tabview.tab("Graph"), text="", font=customtkinter.CTkFont(size=20, weight="bold"))
self.cost_label.grid(row=3, column=2, sticky="nw", pady=(0,20))
self.time_label = customtkinter.CTkLabel(self.tabview.tab("Graph"), text="", font=customtkinter.CTkFont(size=20, weight="bold"))
self.time_label.grid(row=4, column=2, sticky="nw", pady=0)

# Set default values
self.appearance_mode_optionMenu.set("Dark")
self.combobox_1.set("Start")
self.combobox_2.set("Goal")

# Digunakan untuk mengganti mode {Light/Dark}
def change_appearance_mode_event(self, new_appearance_mode: str):
    customtkinter.set_appearance_mode(new_appearance_mode)

# Digunakan untuk menampilkan gambar
def load_image(self, path, image_size):
    image = Image.open(PATH + path).resize((image_size, image_size))
    return customtkinter.CTkImage(image)

```

Gambar 2.3.4 Source Code File main.py

```

# Digunakan untuk mencari map dengan menggunakan file dialog serta beberapa exception
def select_map(self):
    self.file_is_selected = False;
    self.mapName = filedialog.askopenfilename(title="Select a map", filetypes=(("txt files", "*.txt"), ("All Files", "*.*")))
    self.file_name = os.path.basename(self.mapName)
    self.file_ext = os.path.splitext(self.mapName)[1] # Mengambil ekstensi file
    self.combobox_1.configure(values=[""])
    self.combobox_2.configure(values=[""])
    self.error_info.configure(text="")
    try:
        if (self.file_ext==".txt"):
            self.file_is_selected = True;
            self.file_info.configure(text=self.file_name, text_color="green", font=customtkinter.CTkFont(size=15, weight="bold"))
            self.mtr, self.nodes, self.listnodes = p.parse_into_adjacency_mtr(self.mapName)
            self.node_coords = [self.nodes[label] for label in self.listnodes]
            array = []
            for node in self.listnodes:
                array.append(node)
            self.combobox_1.configure(values=array)
            self.combobox_2.configure(values=array)
        else:
            self.file_info.configure(text="Wrong Input File!", text_color="red", font=customtkinter.CTkFont(size=15, weight="bold"))
    except Exception as e:
        err_msg = str(e)
        if (err_msg == "too many values to unpack (expected 3)"):
            self.error_info.configure(text="Cek Ulang txt", text_color="red", font=customtkinter.CTkFont(size=15, weight="bold"))
        else:
            self.error_info.configure(text=err_msg, text_color="red", font=customtkinter.CTkFont(size=15, weight="bold"))

```

Gambar 2.3.5 Source Code File main.py

```

# Digunakan untuk mencari solusi menggunakan algoritma yang ada
def execute(self):
    selected_value = self.radio_var.get()
    value1 = self.combobox_1.get()
    value2 = self.combobox_2.get()
    if (self.file_is_selected==False) :
        self.error_info.configure(text="Insert File!", text_color="red", font=customtkinter.CTkFont(size=15, weight="bold"))
    else :
        self.error_info.configure(text="")
        if (value1 == "Start" or value2 == "Goal"):
            self.error_info.configure(text="Select Nodes Search!", text_color="red", font=customtkinter.CTkFont(size=15, weight="bold"))
        else:
            if (selected_value == 0) :
                self.algorithm_label.configure(text="A* Algorithm Result", text_color="white", font=customtkinter.CTkFont(size=30, weight="bold"))
                self.visualizeGraph()
                self.startTime = time.perf_counter()
                self.path, self.cost = aStar.astar(value1, value2, self.graph, self.nodes, True)
                self.endTime = time.perf_counter()
                self.visualizeInfo()
                self.visualizeTable()
                self.visualizeMap()
            elif (selected_value == 1) :
                self.algorithm_label.configure(text="UCS Algorithm Result", text_color="white", font=customtkinter.CTkFont(size=30, weight="bold"))
                self.visualizeGraph()
                self.startTime = time.perf_counter()
                self.path, self.cost = UCS.ucs(value1, value2, self.graph)
                self.endTime = time.perf_counter()
                self.visualizeInfo()
                self.visualizeTable()
                self.visualizeMap()

```

Gambar 2.3.6 Source Code File main.py

```

# Digunakan untuk memvisualisasikan Graph
def visualizeGraph(self):
    self.graph = p.parse_adjacency_matrix(self.mtr, self.listnodes)
    if hasattr(self, "fig"):
        self.ax.clear()
    else:
        self.fig = plt.figure(figsize=(5, 5))
        self.ax = self.fig.add_subplot(111)
    pos = {label: coord for label, coord in zip(self.listnodes, self.node_coords)}
    nx.draw(self.graph, pos, with_labels=True, ax=self.ax)
    if hasattr(self, "canvas"):
        self.canvas.draw_idle()
    else:
        canvas = FigureCanvasTkAgg(self.fig, master=self.place)
        canvas.draw()
        canvas.get_tk_widget().pack(expand=True)
        self.canvas = canvas

# Digunakan untuk memvisualisasikan Path, Distance, dan Execution Time
def visualizeInfo(self):
    self.result_label.configure(text="Result", text_color="white", font=customtkinter.CTkFont(size=30, weight="bold"))
    self.graph_path_label.configure(text="Path = " + ' - '.join(self.path))
    self.cost_label.configure(text="Total Distance = " + str(self.cost*100) + " km")
    self.time_label.configure(text="Execution Time = " + str((self.endTime-self.startTime)*1000) + " ms")

```

Gambar 2.3.7 Source Code File main.py

```

# Digunakan untuk memvisualisasikan tabel path beserta jarak masing-masing
def visualizeTable(self):
    length = len(self.path)
    arr_distance = [0 for i in range(length-1)]
    for i in range(length-1):
        arr_distance[i] = aStar.find_euclidean_distance(self.nodes[self.path[i]], self.nodes[self.path[i+1]])*100
    arr_path = [0 for i in range(length-1)]
    for i in range(length-1):
        arr_path[i] = self.path[i] + " - " + self.path[i+1]
    tuple_list = list(zip(arr_path, arr_distance))
    style = ttk.Style()
    style.configure('Treeview', font=('TkDefaultFont', 15), rowheight=30, cellwidth=300)
    style.configure('Treeview.Heading', font=('TkDefaultFont', 14, 'bold'))
    self.table = ttk.Treeview(self.tabview.tab("Graph"))
    self.table.grid(row=2, column=2, sticky="nw", pady=(0,20))
    self.table['height'] = len(tuple_list)
    self.table['columns'] = ('Nodes', 'Distance')
    self.table.heading('Nodes', text='Nodes', anchor=tk.CENTER)
    self.table.heading('Distance', text='Distance', anchor=tk.CENTER)
    self.table.column('#0', width=0, stretch=tk.NO)
    self.table.column('Nodes', anchor=tk.CENTER, width=400)
    self.table.column('Distance', anchor=tk.CENTER, width=200)
    data = tuple_list
    for i in range(len(data)):
        self.table.insert('',i, values=data[i])

# Digunakan untuk memvisualisasikan map {Bonus}
def visualizeMap(self):
    self.map_widget = TkinterMapView(self.tabview.tab("Map"), corner_radius=0)
    self.map_widget.grid(row=1, rowspan=1, column=0, columnspan=3, sticky="nswe", padx=(0, 0), pady=(0, 0))
    locations = self.node_coords
    loc_text = self.listnodes
    lat, lng = locations[0][:2]
    self.map_widget.set_position(lat, lng)
    self.map_widget.set_zoom(15)
    for loc in locations:
        lat, lng = loc[0], loc[1]
        marker = self.map_widget.set_marker(lat, lng, text=loc_text[locations.index(loc)])
    for i in range(len(self.path)-1):
        lat, lng = self.nodes[self.path[i]], self.nodes[self.path[i+1]]
        marker = self.map_widget.set_path([lat, lng], color="blue")

if __name__ == "__main__":
    app = PathFinder()
    app.mainloop()

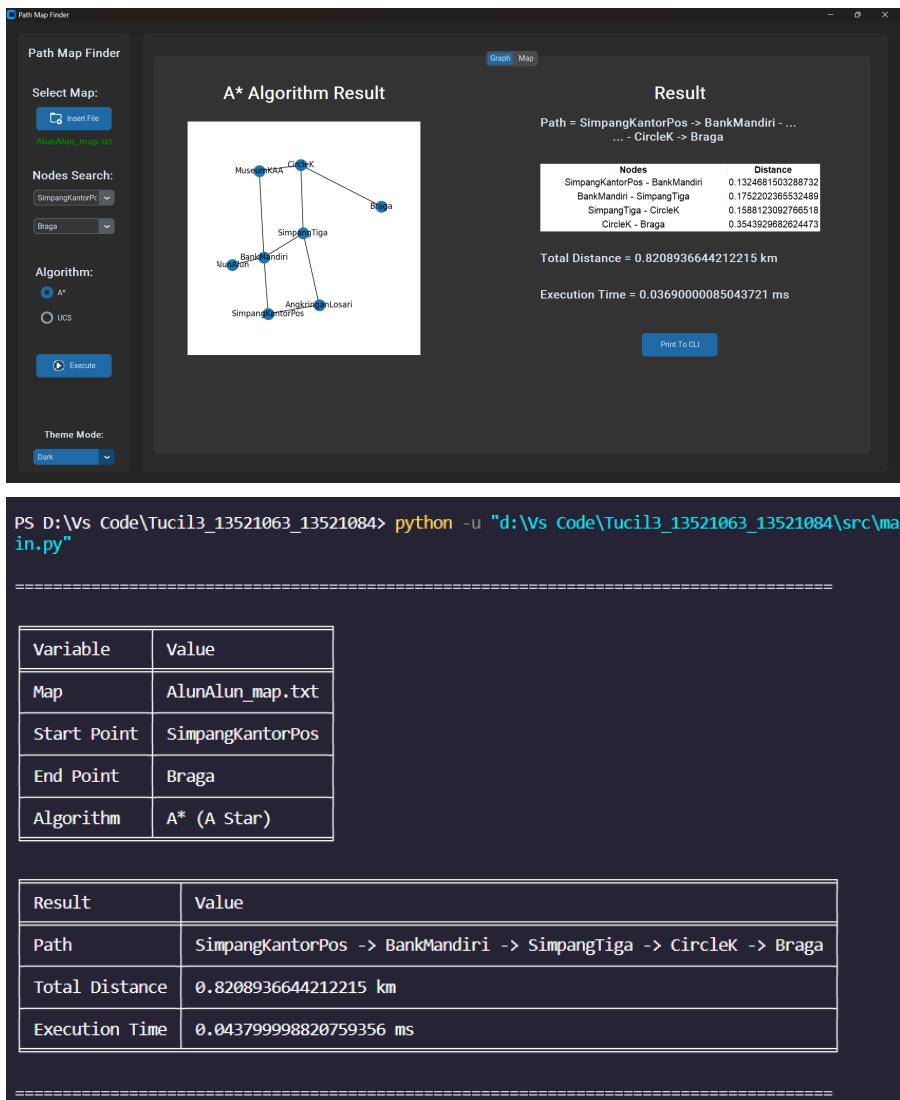
```

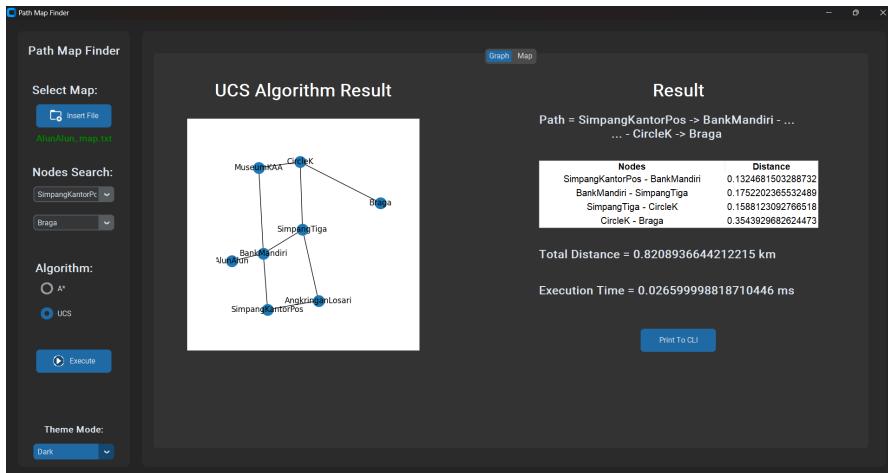
Gambar 2.3.7 Source Code File main.py

BAB III

EKSPERIMENT DAN TESTING

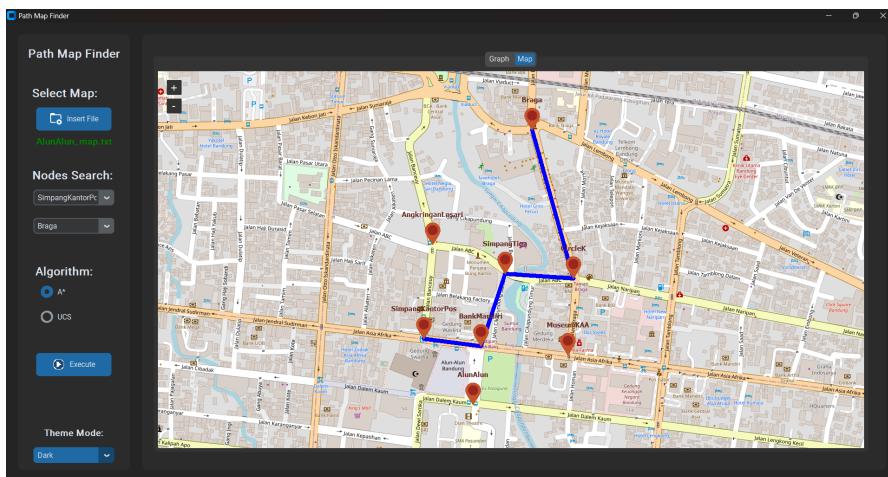
3.1 Testing untuk Peta daerah Alun-Alun Bandung



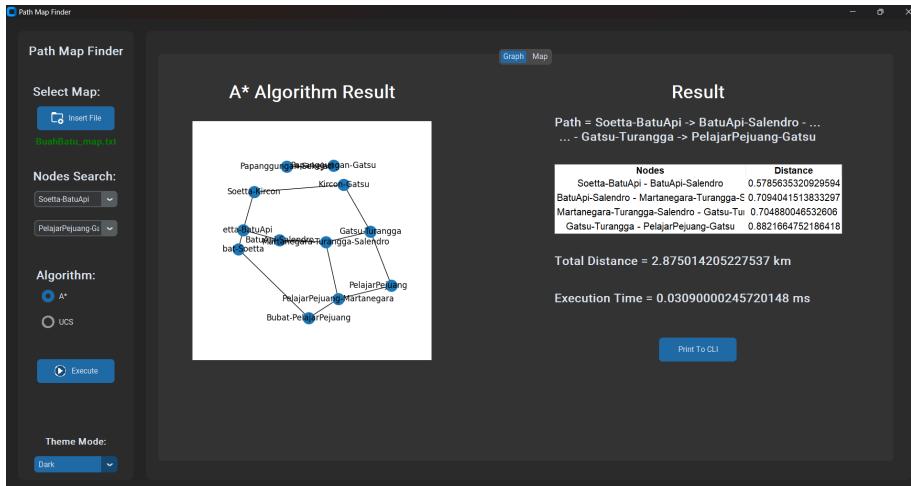


Variable	Value
Map	AlunAlun_map.txt
Start Point	SimpangKantorPos
End Point	Braga
Algorithm	UCS (Uniform cost search)

Result	Value
Path	SimpangKantorPos -> BankMandiri -> SimpangTiga -> CircleK -> Braga
Total Distance	0.8208936644212215 km
Execution Time	0.026599998818710446 ms

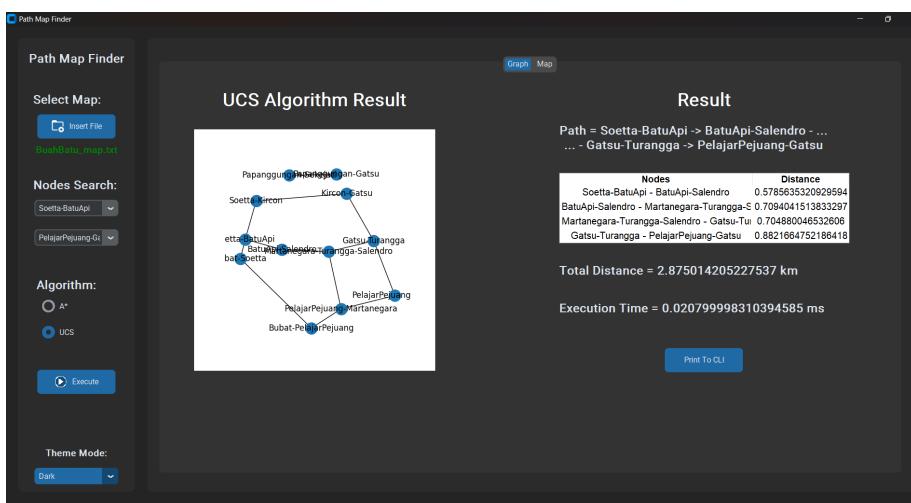


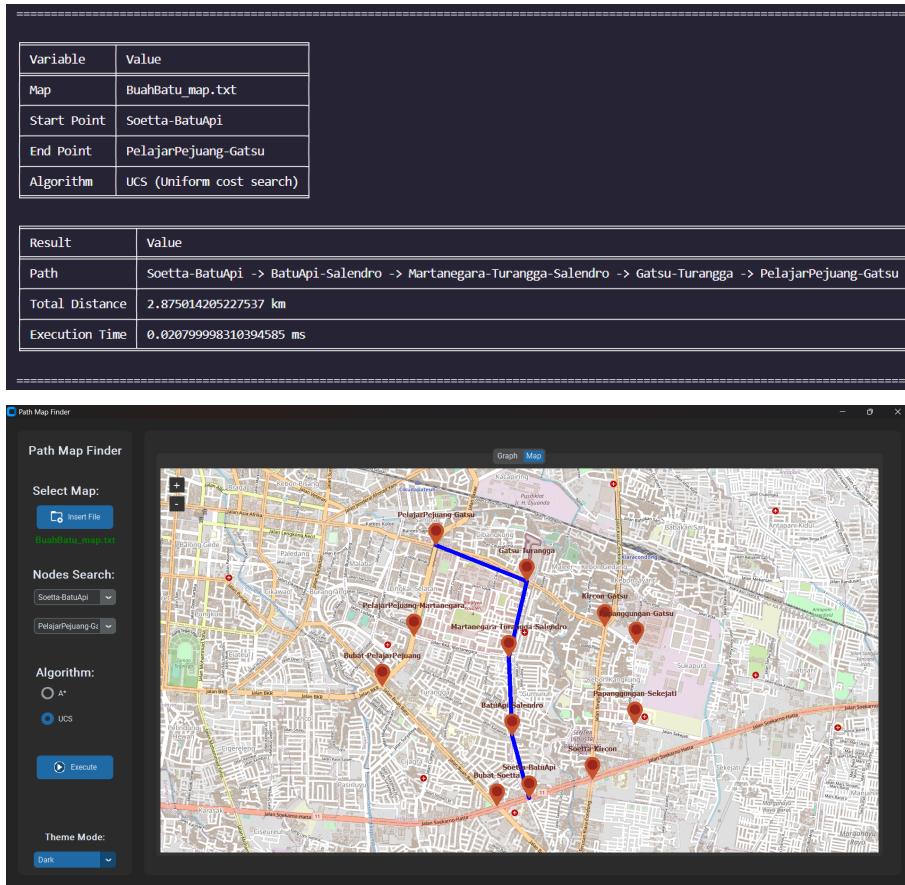
3.2 Testing untuk Peta daerah Buah Batu



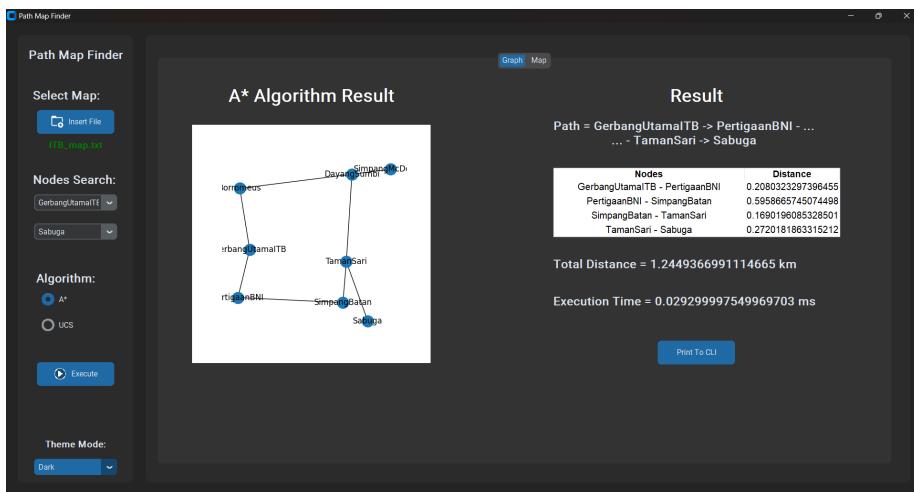
Variable	Value
Map	BuahBatu_map.txt
Start Point	Soetta-BatuApi
End Point	PelajarPejuang-Gatsu
Algorithm	A* (A Star)

Result	Value
Path	Soetta-BatuApi -> BatuApi-Salendro -> Martanegara-Turangga-Salendro -> Gatsu-Turangga -> PelajarPejuang-Gatsu
Total Distance	2.875014205227537 km
Execution Time	0.03090000245720148 ms



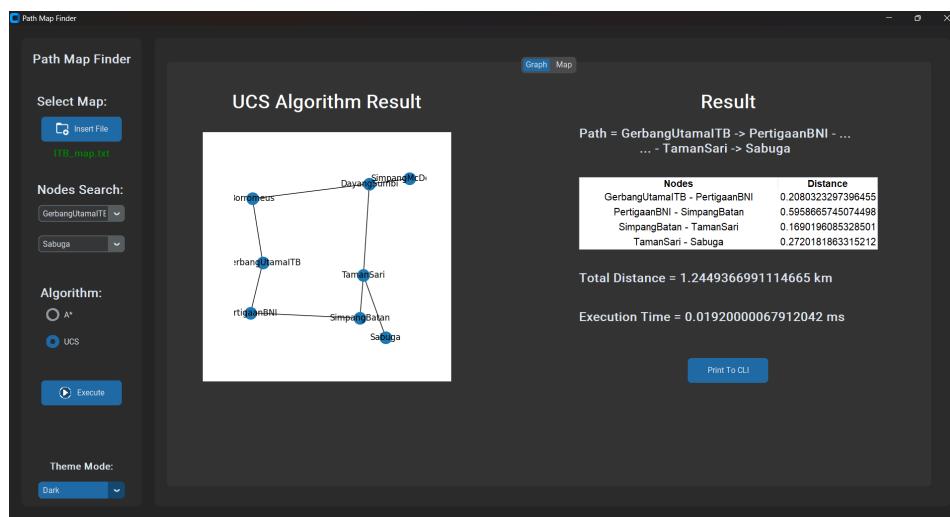


3.3 Testing untuk Peta daerah ITB



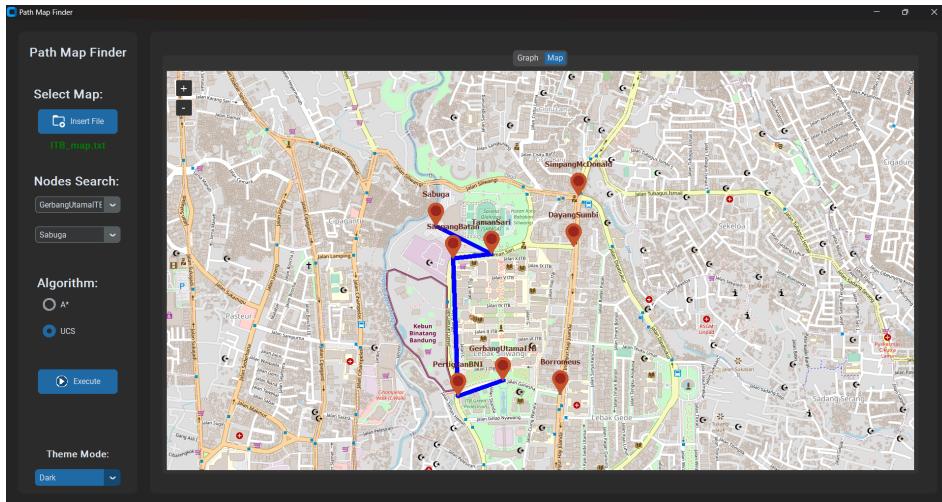
Variable	Value
Map	ITB_map.txt
Start Point	GerbangUtamaITB
End Point	Sabuga
Algorithm	A* (A Star)

Result	Value
Path	GerbangUtamaITB -> PertigaanBNI -> SimpangBatan -> TamanSari -> Sabuga
Total Distance	1.2449366991114665 km
Execution Time	0.029299997549969703 ms

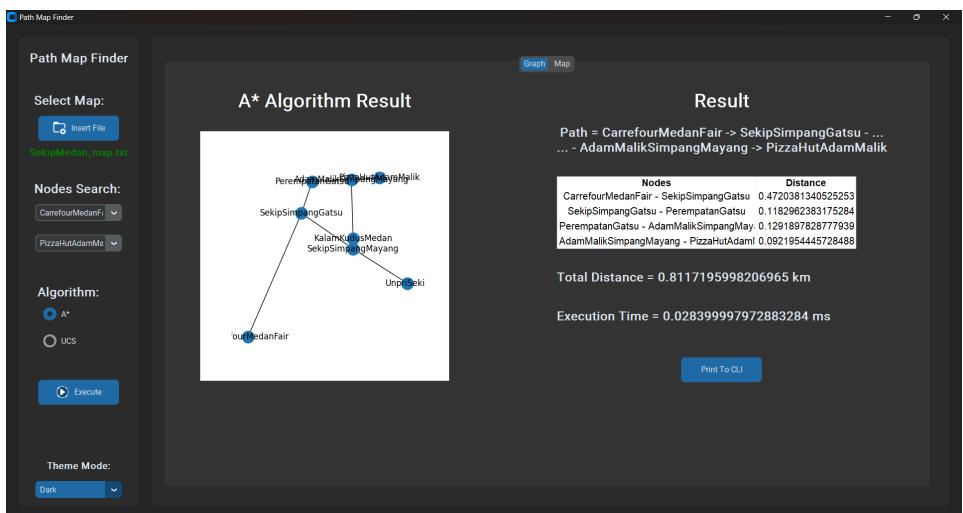


Variable	Value
Map	ITB_map.txt
Start Point	GerbangUtamaITB
End Point	Sabuga
Algorithm	UCS (Uniform cost search)

Result	Value
Path	GerbangUtamaITB -> PertigaanBNI -> SimpangBatan -> TamanSari -> Sabuga
Total Distance	1.2449366991114665 km
Execution Time	0.01920000067912042 ms

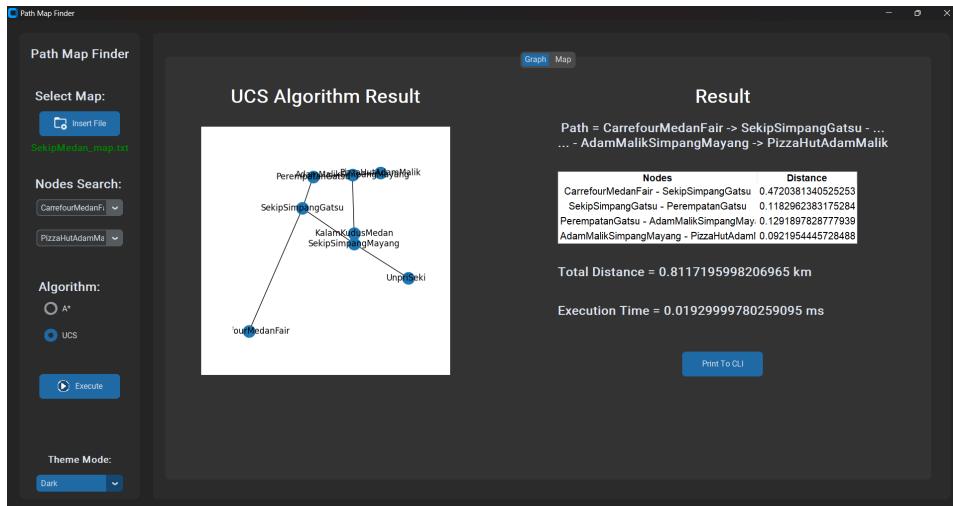


3.4 Testing untuk Peta daerah Ring Road Medan



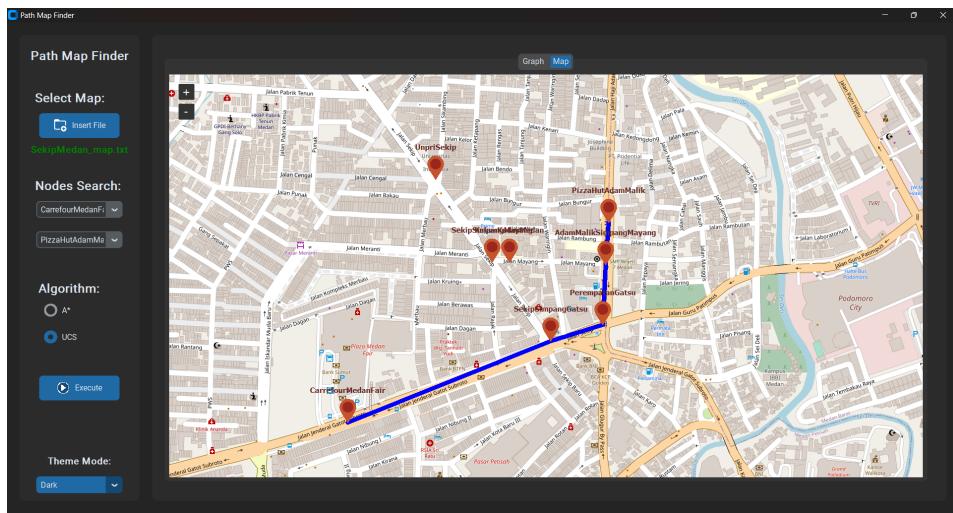
Variable	Value
Map	SekipMedan_map.txt
Start Point	CarrefourMedanFair
End Point	PizzaHutAdamMalik
Algorithm	A* (A Star)

Result	Value
Path	CarrefourMedanFair -> SekipSimpangGatsu -> PerempatanGatsu -> AdamMalikSimpangMayang -> PizzaHutAdamMalik
Total Distance	0.8117195998206965 km
Execution Time	0.028399997972883284 ms



Variable	value
Map	SekipMedan_map.txt
Start Point	CarrefourMedanFair
End Point	PizzaHutAdamMalik
Algorithm	UCS (Uniform cost search)

Result	value
Path	CarrefourMedanFair -> SekipSimpangGatsu -> PerempatanGatsu -> AdamMalikSimpangMayang -> PizzaHutAdamMalik
Total Distance	0.8117195998206965 km
Execution Time	0.01929999780259095 ms



DAFTAR PUSTAKA

Munir, R. (2023). Algoritma Penentuan Rute (*Route / Path Planning*) (Bagian 2). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023. Diakses pada 7 April 2023, pukul 20.43 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>

LAMPIRAN

Link Repository Program

https://github.com/AustinPardosi/Tucil3_13521063_13521084

Checklist Tugas Kecil

Poin	Ya	Tidak
1. Program dapat menerima input graf	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program dapat menghitung lintasan terpendek dengan UCS	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Program dapat menghitung lintasan terpendek dengan A*	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Program dapat menampilkan lintasan terpendek serta jaraknya	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	<input checked="" type="checkbox"/>	<input type="checkbox"/>