Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Dibuat oleh:

Margaretha Olivia Haryono	13521071
Nathan Tenka	13521172

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

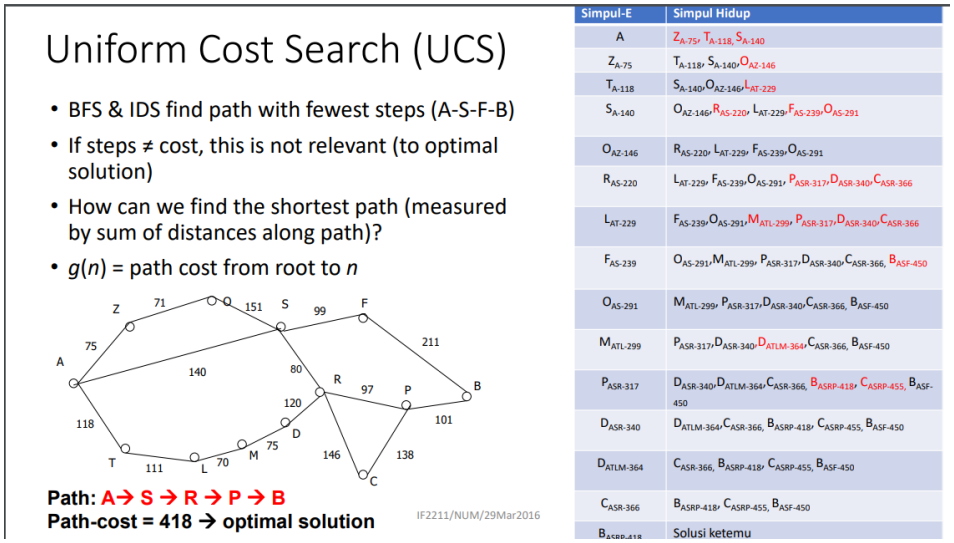
INSTITUT TEKNOLOGI BANDUNG

TAHUN AJARAN 2022/2023

Bab I

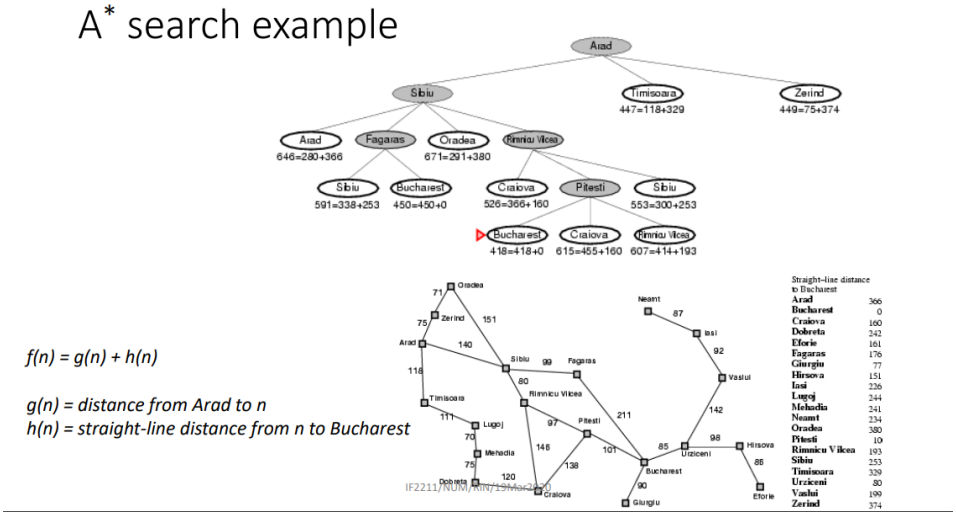
Deskripsi Persoalan

UCS dan A* adalah dua algoritma pencarian rute yang umum diketahui. UCS mencari jalur dengan mengambil simpul yang jaraknya dari titik mulai paling kecil sampai ditemukan simpul tujuan. Sedangkan A* menggunakan heuristik tambahan yang *admissible* (tidak pernah memperkirakan nilai yang lebih besar dari sebenarnya) dan mengambil simpul dengan jarak dari akar dan jarak titik tujuan (berdasarkan heuristik) yang minimum lebih dulu.



Contoh algoritma UCS

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf



Contoh algoritma A*

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

Pada tugas kali ini penulis diminta untuk membuat program pencarian rute berdasarkan *google map* di sekitar kota Bandung. Pencarian rute dilakukan menggunakan algoritma UCS dan A*. Peta direpresentasikan dengan graf yang dibuat dalam bentuk matriks ketetanggan. Program bisa menerima simpul asal dan tujuan, lalu menentukan jalur terpendek antara kedua simpul tersebut. Jalur kemudian ditunjukkan di peta/graf. Nilai heuristik yang digunakan adalah jarak garis lurus.

Bab II

Kode Program

main.py

```
import tkinter as tk
from tkinter import *
from tkinter.ttk import *
from tkinter.filedialog import askopenfile
import tkintermapview as tkMap
import customtkinter as ctk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
import numpy as np
import networkx as nx
from pathlib import Path
import UCS
import Utils
import Map
import AStar

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

startCoord = None
endCoord = None
startMarker = None
endMarker = None
resultPath = None
m = None
coord = None
names = None

root = ctk.CTk()  # create root window
root.title("PathFinder A* & UCS")  # title of the GUI window
root.geometry(f"{1050}x{650}")  # specify the max size the window can expand to
```

```
searchModeMap = IntVar()
searchModeGraph = IntVar()

startNodeGraph = StringVar()
endNodeGraph = StringVar()

def openFileGraph() :
    global m,coord, names
    file_path = askopenfile(mode='r', filetypes=[('text files', '*.txt')])
    if file_path is not None:
        fName = Path(file_path.name)
        filename.configure(text="File name : " + fName.name)
        searchGraphBtn.configure(state=NORMAL)
        try:
            m, coord, names = Utils.Util.readMatrix(file_path.name)
            plotGraph(m,coord,names)
            nodeList = []
            for i in range(len(names)):
                nodeList.append(names[i])
            startCbBox.configure(values=nodeList)
            endCbBox.configure(values=nodeList)
            errorLabel.configure(text="")
        except:
            errorLabel.configure(text="input not valid")

def searchGraph() :
    try :
        if(searchModeGraph.get() == 1) : # UCS
            shortest_route, shortest_dist =
UCS.UCS.searchPath(names.index(startNodeGraph.get()),names.index(endNodeGraph.get()),m)
        elif (searchModeGraph.get() == 2) : # AStar
            shortest_route, shortest_dist =
AStar.AStar.searchPath(names.index(startNodeGraph.get()),names.index(endNodeGraph.get()),m,coord)
        route_names = []
        for n in shortest_route :
            route_names.append(names[n])
        plotGraph(m,coord,names,route_names)
        dist_lbl_graph.configure(text="Shortest distance : " + str(shortest_dist) + " meter(s)")
```

```
        path = ""
        for i in range(len(route_names)) :
            if (i < len(route_names)-1) :
                path += str(route_names[i]) + " - "
            else :
                path += str(route_names[i])
        path_lbl_graph.configure(text="Path result : " + path, text_color="white")
    except :
        dist_lbl_graph.configure(text="Shortest distance : ")
        path_lbl_graph.configure(text="Path result : Path not found", text_color="red")

# Map functions
def drawGraphOnMap(coordList, nameList, edgeList) :
    # Menggambar marker berdasarkan list koordinat
    map_widget.delete_all_path()
    map_widget.delete_all_marker()
    count = 1
    for i in range (len(coordList)) :
        map_widget.set_marker(coordList[i][1], coordList[i][0], text=(nameList[i] if nameList[i] != "" else
"Node "+str(count)))
        count += 1
    for e in edgeList :
        map_widget.set_path(e, color="blue", width=5)

def openFileMap() :
    map_path = askopenfile(mode='r', filetypes=[('text files', '*.txt')])
    if (map_path is not None) :
        center, coordList, names, edgeList = Map.MapPathSearch.saveGraphFile(map_path.name)
        map_widget.set_position(center[1],center[0])
        msgMapLbl.configure(text="Map uploaded", text_color="white")
        msgMapLbl.place(relx=0.53,rely=0.72)
        drawGraphOnMap(coordList,names,edgeList)

def add_start_coord(coords):
    # Menambahkan titik awal pada peta
    global startCoord, startMarker
    startCoord = (coords[1],coords[0])
    if (resultPath != None) :
```

```
        resultPath.delete()
    if (endCoord != None) :
        searchMapBtn.configure(state=NORMAL)
    if (startMarker != None) :
        startMarker.delete()
    # Menampilkan alamat pada koordinat yang ditunjuk
    addr = tkMap.convert_coordinates_to_address(coords[0],coords[1])
    displayAddr = ""
    if (addr.street != None) :
        displayAddr += addr.street
    if (addr.city != None and displayAddr != "") :
        displayAddr += ", " + addr.city
    startMarker = map_widget.set_marker(coords[0], coords[1], text=((displayAddr) if displayAddr != ""
else "Start"))

def add_end_coord(coords):
    # Menambahkan titik akhir pada peta
    global endCoord, endMarker
    endCoord = (coords[1],coords[0])
    if (resultPath != None) :
        resultPath.delete()
    if (startCoord != None) :
        searchMapBtn.configure(state=NORMAL)
    if (endMarker != None) :
        endMarker.delete()
    # Menampilkan alamat pada koordinat yang ditunjuk
    addr = tkMap.convert_coordinates_to_address(coords[0],coords[1])
    displayAddr = ""
    if (addr.street != None) :
        displayAddr += addr.street
    if (addr.city != None and displayAddr != "") :
        displayAddr += ", " + addr.city
    endMarker = map_widget.set_marker(coords[0], coords[1], text=((displayAddr) if displayAddr != ""
else "End"))

def search_path_map() :
    global resultPath
    # Melakukan pencarian jalur pada peta
```

```
    if (resultPath != None) :
        resultPath.delete()
    msgMapLbl.place_forget()
    if (searchModeMap.get() == 1) :
        method = 'UCS'
    elif (searchModeMap.get() == 2) :
        method = 'A*'
    else :
        msgMapLbl.configure(text="Select a search method first !", text_color="red")
        msgMapLbl.place(relx=0.53,rely=0.72)
        return
    shortest_route, shortest_distance, graph, success =
Map.MapPathSearch.search(startCoord,endCoord,method)
    if (success) :
        if (shortest_distance > 0) :
            shortest_route_coor = Map.MapPathSearch.convertPathToCoorPath(shortest_route,graph)
            map_widget.set_position(startCoord[1],startCoord[0])
            map_widget.set_zoom(15)
            dist_lbl.configure(text="Shortest distance : " + str(shortest_distance) + " meter(s)")
            resultPath = map_widget.set_path(shortest_route_coor,color="red")
        else :
            dist_lbl.configure(text="Shortest distance : ")
            msgMapLbl.configure(text="Start and end is on the same node", text_color="red")
            msgMapLbl.place(relx=0.53,rely=0.72)
    else :
        dist_lbl.configure(text="Shortest distance : ")
        msgMapLbl.configure(text="Route not found", text_color="red")
        msgMapLbl.place(relx=0.53,rely=0.72)

def search_loc_map() :
    # Memindahkan posisi peta ke lokasi yang dicari
    msgMapLbl.place_forget()
    # Diasumsikan pindah ke tempat yang relatif jauh, jadi semua path dan marker dihapus
    map_widget.delete_all_path()
    map_widget.delete_all_marker()
    newCoord = tkMap.convert_address_to_coordinates(locTxtBox.get('0.0',END))
    if (newCoord != None) :
        map_widget.set_position(newCoord[0],newCoord[1])
```

```
        map_widget.set_zoom(15)
    else :
        msgMapLbl.configure(text="Location not found", text_color="red")
        msgMapLbl.place(relx=0.53,rely=0.72)

def plotGraph(m, coord, names, path=[]):
    # Menggambar graf di tempat yang disediakan
    adj_matrix = np.array(m)

    # Create a directed weighted graph from the weighted directed adjacency matrix
    graph = nx.Graph(adj_matrix)

    # Convert 2D matrix to dictionary of node positions
    pos = {names[i]: tuple(coord[i]) for i in range(len(coord))}

    nodes_name_mapping = {i: names[i] for i in range(len(names))}
    graph = nx.relabel_nodes(graph, nodes_name_mapping)
    # pos = {nodes_name_mapping[k]: pos[k] for k in nodes_name_mapping}

    path_edges = []
    if (len(path) > 0) :
        for i in range(len(path)-1):
            path_edges.append(tuple((path[i], path[i+1])))

    a = fig.add_subplot(111)
    # Draw graph with fixed node positions and edge labels
    nx.draw_networkx(graph, pos=pos, ax=a, with_labels=True, node_color=['blue' if e in path else
'lightblue' for e in graph.nodes()],
                     node_size=500, font_size=14, font_weight='bold',
                     edge_color=['red' if e in path_edges else 'black' for e in graph.edges()])
    # edge_labels = nx.get_edge_attributes(self.graph, "weight")
    # nx.draw_networkx_edge_labels(self.graph, pos, edge_labels, rotate=False)
    plt.axis("off")
    canvas.draw()

# Title label
title = ctk.CTkLabel(root,text="A* and UCS Pathfinder",font=('Arial',25))
title.place(relx=0.37,rely=0.05)
```

```
# Graph display
graphFrame = ctk.CTkFrame(root, width=600, height=350, bg_color="transparent")
graphFrame.place(relx=0.05,rely=0.15)
fig = plt.Figure(figsize=(6, 3.58), dpi=100)
canvas = FigureCanvasTkAgg(fig, graphFrame)
toolbar = NavigationToolbar2Tk(canvas,graphFrame,pack_toolbar=False)
toolbar.update()
toolbar.pack(side=tk.BOTTOM, fill=tk.X)
canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Upload graph file button and labels
filename = ctk.CTkLabel(root,text="File name :")
filename.place(relx=0.1,rely=0.62)
dist_lbl_graph = ctk.CTkLabel(root,text="Shortest distance :")
dist_lbl_graph.place(relx=0.1,rely=0.65)
path_lbl_graph = ctk.CTkLabel(root,text="Path :")
path_lbl_graph.place(relx=0.1,rely=0.68)
uploadGraphBtn = ctk.CTkButton(root,text="Upload Graph",command=openFileGraph)
uploadGraphBtn.place(relx=0.16,rely=0.73)

errorLabel = ctk.CTkLabel(root,text="")
errorLabel.place(relx=0.4,rely=0.68)

# Graph start node and end node select
startCbBox = ctk.CTkComboBox(root, values=[''], variable=startNodeGraph, width=100)
startCbBox.set("Start Node")
startCbBox.place(relx=0.12,rely=0.78)
endCbBox = ctk.CTkComboBox(root, values=[''], variable=endNodeGraph, width=100)
endCbBox.set("End Node")
endCbBox.place(relx=0.24,rely=0.78)

# Search graph button
searchGraphBtn = ctk.CTkButton(root,text="Search",command=searchGraph,state="disabled")
searchGraphBtn.place(relx=0.16,rely=0.83)

# Search mode select button graph
UCSGraphBtn = ctk.CTkRadioButton(root,text="UCS", variable=searchModeGraph, value=1)
```

```
UCSGraphBtn.place(relx=0.18,rely=0.88)
AStarMapBtn = ctk.CTkRadioButton(root,text="A*", variable=searchModeGraph, value=2)
AStarMapBtn.place(relx=0.18,rely=0.92)

# Search location textbox and button
locTxtBox = ctk.CTkTextbox(root,width=300,height=30,activate_scrollbars=False)
locTxtBox.place(relx=0.63,rely=0.67)
searchLocBtn = ctk.CTkButton(root,text="Search
Location",width=40,height=30,command=search_loc_map)
searchLocBtn.place(relx=0.53,rely=0.67)

# Error label
msgMapLbl = ctk.CTkLabel(root,height=10)

# Upload map button
uploadMapBtn = ctk.CTkButton(root, text="Upload Map", command=openFileMap)
uploadMapBtn.place(relx=0.65,rely=0.75)

# Search Map button
searchMapBtn = ctk.CTkButton(root,text="Search Path",command=search_path_map,state="disabled")
searchMapBtn.place(relx=0.65,rely=0.8)

# Search mode select button map
UCSMapBtn = ctk.CTkRadioButton(root,text="UCS", variable=searchModeMap, value=1)
UCSMapBtn.place(relx=0.665,rely=0.85)
AStarMapBtn = ctk.CTkRadioButton(root,text="A*", variable=searchModeMap, value=2)
AStarMapBtn.place(relx=0.665,rely=0.9)

# Map widget
map_widget = tkMap.TkinterMapView(root, width=600, height=400, corner_radius=0)
map_widget.place(relx=0.5,rely=0.15)
dist_lbl = ctk.CTkLabel(root,text="Shortest distance :")
dist_lbl.place(relx=0.53,rely=0.62)

map_widget.add_right_click_menu_command(label="Add start point",
                                        command=add_start_coord,
                                        pass_coords=True)
map_widget.add_right_click_menu_command(label="Add end point",
```

```
                                command=add_end_coord,
                                pass_coords=True)

root.protocol("WM_DELETE_WINDOW", root.quit)
root.mainloop()
```

AStar.py

```
from Utils import Node
import math
import osmnx as ox

class AStar:
    @staticmethod
    def searchPath(startNodeNum, destNodeNum, matrix, coord):
        # AStar untuk matriks ketetanggaan
        liveNodes = []
        visitedNodesNum = []
        startCoord = coord[startNodeNum]
        destCoord = coord[destNodeNum]
        currNode = Node(startNodeNum, 0, [startNodeNum], AStar.euc_dist(startCoord, destCoord)) # Simpul
ekspan
        while (currNode.number != destNodeNum):
            visitedNodesNum.append(currNode.number)
            # Masukkan tetangga ke daftar simpul hidup
            for i in range(len(matrix[currNode.number])):
                # Hanya masukkan simpul yang belum pernah dikunjungi ke daftar simpul hidup
                if ((i not in visitedNodesNum) and matrix[currNode.number][i] > 0):
                    liveNodes.append(Node(i, currNode.distFromRoot+matrix[currNode.number][i],
                                          currNode.prevPath + [i], AStar.euc_dist(coord[i], destCoord)))
            if (len(liveNodes) == 0) : # tidak ada tetangga yang bisa dikunjungi lagi
                return
            liveNodes.sort(key=lambda x : x.distFromRoot+x.distFromDest)
            currNode = liveNodes[0]
            liveNodes.remove(currNode)
        return currNode.prevPath, currNode.distFromRoot

    @staticmethod
    def euc_dist(point1, point2):
```

```
        return math.sqrt((point2[0]-point1[0])**2 + (point2[1]-point1[1])**2)

class AStarOSMNX :
    @staticmethod
    def searchPath(startNode, destNode, graph):
        # AStar untuk graf osmnx
        # Mencari jalur terpendek berdasarkan panjang
        liveNodes = []
        visitedNodes= []
        nodes,edges = ox.graph_to_gdfs(graph)
        coord = (nodes['x'], nodes['y'])
        startCoord = (coord[0].loc[startNode], coord[1].loc[startNode])
        destCoord = (coord[0].loc[destNode], coord[1].loc[destNode])
        currNode = Node(startNode, 0, [startNode], AStar.euc_dist(startCoord, destCoord)) # Simpul ekspan
        while (currNode.number != destNode):
            visitedNodes.append(currNode.number)
            # Cari semua tetangga simpul ekspan
            try :
                neighbour = edges['length'].loc[currNode.number]
            except : # Jika tidak ada tetangga, lanjut
                if (len(liveNodes) == 0) : # Jika sudah tidak ada simpul hidup dan belum sampai tujuan, berhenti
                    return
                currNode = liveNodes[0]
                liveNodes.remove(currNode)
                continue
            # Masukkan tetangga ke daftar simpul hidup
            for idx,l in neighbour.items() :
                if ((idx[0]) not in set(visitedNodes)) :
                    liveNodes.append(Node(idx[0], currNode.distFromRoot+l,
                                          currNode.prevPath + [idx[0]],
                                          AStar.euc_dist((coord[0].loc[idx[0]],coord[1].loc[idx[0]]), destCoord)))
            if (len(liveNodes) == 0) : # tidak ada tetangga yang bisa dikunjungi lagi
                return
            liveNodes.sort(key=lambda x : x.distFromRoot+x.distFromDest)
            currNode = liveNodes[0]
            liveNodes.remove(currNode)
        return currNode.prevPath, currNode.distFromRoot
```

UCS.py

```
from Utils import Node
```

```
import osmnx as ox

class UCS :
    @staticmethod
    def searchPath(startNodeNum, destNodeNum, matrix) :
        # Route searching pada matriks ketetanggaan
        liveNodes = []
        visitedNodesNum = []
        currNode = Node(startNodeNum,0,[startNodeNum])          # Simpul ekspan
        while (currNode.number != destNodeNum) :
            visitedNodesNum.append(currNode.number)
            for i in range(len(matrix[currNode.number])) : # Masukkan tetangga ke daftar simpul hidup
                if ((i not in visitedNodesNum) and matrix[currNode.number][i] > 0) : # Hanya masukkan simpul
yang belum pernah dikunjungi ke daftar simpul hidup
                    liveNodes.append(Node(i, currNode.distFromRoot + matrix[currNode.number][i],
                                          currNode.prevPath + [i]))
            if (len(liveNodes) == 0) : # tidak ada tetangga yang bisa dikunjungi lagi
                return
            liveNodes.sort(key=lambda x : x.distFromRoot)
            currNode = liveNodes[0]
            liveNodes.remove(currNode)
        return currNode.prevPath, currNode.distFromRoot

class UCSOSMNX :
    @staticmethod
    def searchPath(startNode,destNode,graph) :
        # Route searching pada graf osmnx
        # Mencari jalur terpendek berdasarkan panjang jalan
        if (startNode == destNode) : # Jika simpul mulai sama dengan simpul tujuan
            return [startNode],0
        liveNodes = []
        visitedNodes = []
        currNode = Node(startNode,0,[startNode])
        edges = ox.graph_to_gdfs(graph,nodes=False)
        while (currNode.number != destNode) :
            visitedNodes.append(currNode.number)
            # Cari semua tetangga simpul ekspan
            try :
                neighbour = edges['length'].loc[currNode.number]
```

```
        except : # Jika tidak ada tetangga, lanjut ke simpul hidup berikutnya
            if (len(liveNodes) == 0) : # Jika sudah tidak ada simpul hidup dan belum sampai tujuan, berhenti
                return
            currNode = liveNodes[0]
            liveNodes.remove(currNode)
            continue
        for idx,l in neighbour.items() :
            if ((idx[0]) not in set(visitedNodes)) :
                liveNodes.append(Node(idx[0],currNode.distFromRoot + l,
                                currNode.prevPath + [idx[0]]))
        if (len(liveNodes) == 0) :
            return
        liveNodes.sort(key=lambda x : x.distFromRoot)
        currNode = liveNodes[0]
        liveNodes.remove(currNode)
    return currNode.prevPath, currNode.distFromRoot
```

Map.py

```
import osmnx as ox
import networkx as nx
import numpy as np
import UCS
import Utils
import AStar

ox.settings.use_cache=False

class MapPathSearch :
    savedGraph = None
    boundBox = []
    mat = None
    graphCoords = None

    def saveGraphFile(fileName) :
        # Membaca graf peta dari file txt berisi matriks ketetanggan, koordinat, dan nama
        MapPathSearch.mat,MapPathSearch.graphCoords,names = Utils.Util.readMatrix(fileName)
        adj_matrix = np.array(MapPathSearch.mat)
        MapPathSearch.savedGraph = nx.Graph(adj_matrix)
        nx.set_node_attributes(MapPathSearch.savedGraph, {"x" : 0, "y" : 0})
        for i in range(len(MapPathSearch.savedGraph.nodes)) :
```

```
            MapPathSearch.savedGraph.nodes[i]["x"] = MapPathSearch.graphCoords[i][0]
            MapPathSearch.savedGraph.nodes[i]["y"] = MapPathSearch.graphCoords[i][1]
        edgeList = [] # Penampung koordinat tiap rusuk
        for u,v in MapPathSearch.savedGraph.edges :
            edgeList.append([(MapPathSearch.savedGraph.nodes[u]["y"],
MapPathSearch.savedGraph.nodes[u]["x"]),
                             (MapPathSearch.savedGraph.nodes[v]["y"], MapPathSearch.savedGraph.nodes[v]["x"])])
        minX = MapPathSearch.graphCoords[0][0]
        minY = MapPathSearch.graphCoords[0][1]
        maxX = MapPathSearch.graphCoords[0][0]
        maxY = MapPathSearch.graphCoords[0][1]
        for c in MapPathSearch.graphCoords :
            if (c[0] < minX) :
                minX = c[0]
            elif (c[0] > maxX) :
                maxX = c[0]
            if (c[1] < minY) :
                minY = c[1]
            elif (c[1] > maxY) :
                maxY = c[1]
        MapPathSearch.boundBox = [minX, minY, maxX, maxY]
        center = ((MapPathSearch.boundBox[0] + MapPathSearch.boundBox[2])/2,
(MapPathSearch.boundBox[1] + MapPathSearch.boundBox[3])/2)
        return center, MapPathSearch.graphCoords, names, edgeList

    def saveGraphPoint(start, end) :
        # Download graph berdasarkan bounding box start,end
        dist = Utils.Util.eucliDistanceMap(start[0],start[1],end[0],end[1])
        if (dist > 1000) : # Jika jarak > 1 km, cari jalur yang bisa dilalui kendaraan saja
            MapPathSearch.savedGraph = ox.graph_from_bbox(max(start[1],end[1]) + 0.001,
min(start[1],end[1])-0.001, max(start[0],end[0]) + 0.001, min(start[0],end[0])-0.001,
                                                network_type='drive')
        else :
            MapPathSearch.savedGraph = ox.graph_from_bbox(max(start[1],end[1]) + 0.001,
min(start[1],end[1])-0.001, max(start[0],end[0]) + 0.001, min(start[0],end[0])-0.001,
                                                network_type='walk')
        MapPathSearch.boundBox = ox.graph_to_gdfs(MapPathSearch.savedGraph, edges=False).total_bounds

    def convertPathToCoorPath(nodePath,graph) :
```

```
    # Mengubah path node menjadi path koordinat
    coorPath = []
    try :
        # Graf osmnx
        nodes = ox.graph_to_gdfs(graph,edges=False)
        for n in nodePath :
            nodeData = nodes.loc[n]
            coorPath.append((nodeData['y'],nodeData['x']))
    except :
        # Graf nx
        for n in nodePath :
            coorPath.append((graph.nodes[n]['y'],graph.nodes[n]['x']))
    return coorPath

def searchClosestNode(graph,coor) :
    # Mencari koordinat terdekat untuk graf nx biasa
    if (graph == None) :
        return
    closestNode = graph.nodes[0]
    closestIdx = 0
    for i in range(len(graph.nodes)) :
        if (Utils.Util.eucliDistanceMap(graph.nodes[i]['x'], graph.nodes[i]['y'],coor[0],coor[1]) <
            Utils.Util.eucliDistanceMap(closestNode['x'],closestNode['y'],coor[0],coor[1])) :
            closestNode = graph.nodes[i]
            closestIdx = i
    return closestIdx

def isInBoundBox(coor) :
    # Memeriksa apakah coor ada di dalam bounding box
    if (len(MapPathSearch.boundBox) == 0) :
        return False
    elif (coor[0] < MapPathSearch.boundBox[0] or coor[0] > MapPathSearch.boundBox[2]
        or coor[1] < MapPathSearch.boundBox[1] or coor[1] > MapPathSearch.boundBox[3]) :
        return False
    return True

def search(startCoor, endCoor, method) :
    # Mengembalikan list koordinat nodes dan graf (untuk keperluan fungsi lain)
    if (not(MapPathSearch.isInBoundBox(startCoor)) or not(MapPathSearch.isInBoundBox(endCoor))) :
```

```
        # Jika koordinat ada di luar graf, buat graf baru
        MapPathSearch.saveGraphPoint(startCoor, endCoor)
    try :
        isOSMGraph = True
        startNode = ox.distance.nearest_nodes(MapPathSearch.savedGraph,startCoor[0],startCoor[1])
        endNode = ox.distance.nearest_nodes(MapPathSearch.savedGraph,endCoor[0],endCoor[1])
    except :
        isOSMGraph = False
        startNode = MapPathSearch.searchClosestNode(MapPathSearch.savedGraph,startCoor)
        endNode = MapPathSearch.searchClosestNode(MapPathSearch.savedGraph,endCoor)
    try :
        if (isOSMGraph) :
            if (method == 'UCS') :
                shortest_route, shortest_distance = UCS.UCSOSMNX.searchPath(startNode, endNode,
MapPathSearch.savedGraph)
            elif(method == 'A*') :
                shortest_route, shortest_distance = AStar.AStarOSMNX.searchPath(startNode, endNode,
MapPathSearch.savedGraph)
        else : # Graf nx biasa
            if (method == 'UCS') :
                shortest_route, shortest_distance = UCS.UCS.searchPath(startNode, endNode,
MapPathSearch.mat)
            elif(method == 'A*') :
                shortest_route, shortest_distance = AStar.AStar.searchPath(startNode, endNode,
MapPathSearch.mat, MapPathSearch.graphCoords)
    except : # Gagal
        return -1,-1,None,False
    return shortest_route, shortest_distance, MapPathSearch.savedGraph, True
```

Utils.py

```
import math

# Representasi simpul yang berisi nomor simpul, nama simpul, dan jalur untuk mencapai simpul tersebut
class Node :
  def __init__(self,number,distFromRoot,prevPath,distFromDest=0) :
    self.number = number
    self.distFromRoot = distFromRoot
    self.distFromDest = distFromDest
    self.prevPath = prevPath
```

```
# Fungsi-fungsi yang berhubungan dengan graf
class Util:
    def readMatrix(fileName):
        # Membaca matriks ketetanggan dan koordinat tiap simpul dari file
        f = open(fileName, "r")
        total_lines = len(f.readlines())
        total_nodes = total_lines // 3
        m = []
        coord = []
        nodes_name = []

        f.seek(0)
        lines = f.readlines()
        for i in range(total_nodes):
            m.append([float(x) for x in lines[i].split(' ')])

        for i in range(total_nodes, total_nodes*2):
            coord.append([float(x) for x in lines[i].split(' ')])

        for i in range(total_nodes*2, total_nodes*3):
            nodes_name.append(lines[i].strip())

        for i in range(len(m)):
            for j in range(len(m)):
                if m[i][j] != m[j][i]:
                    raise Exception

        f.close()
        return m, coord, nodes_name

    def eucliDistanceMap(x1,y1,x2,y2) :
        return math.sqrt(pow(x2*111139-x1*111139,2) + pow(y2*111139-y1*111139,2))
```

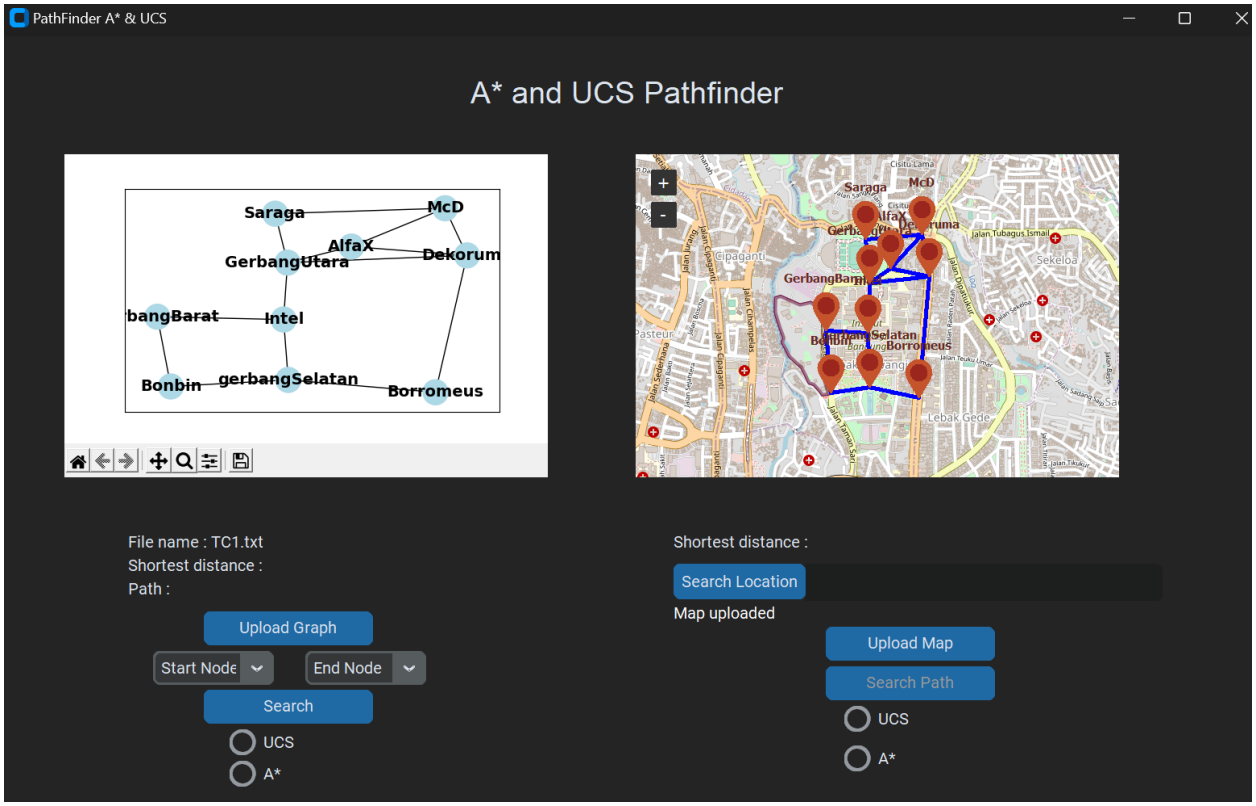
Bab III

Hasil Uji

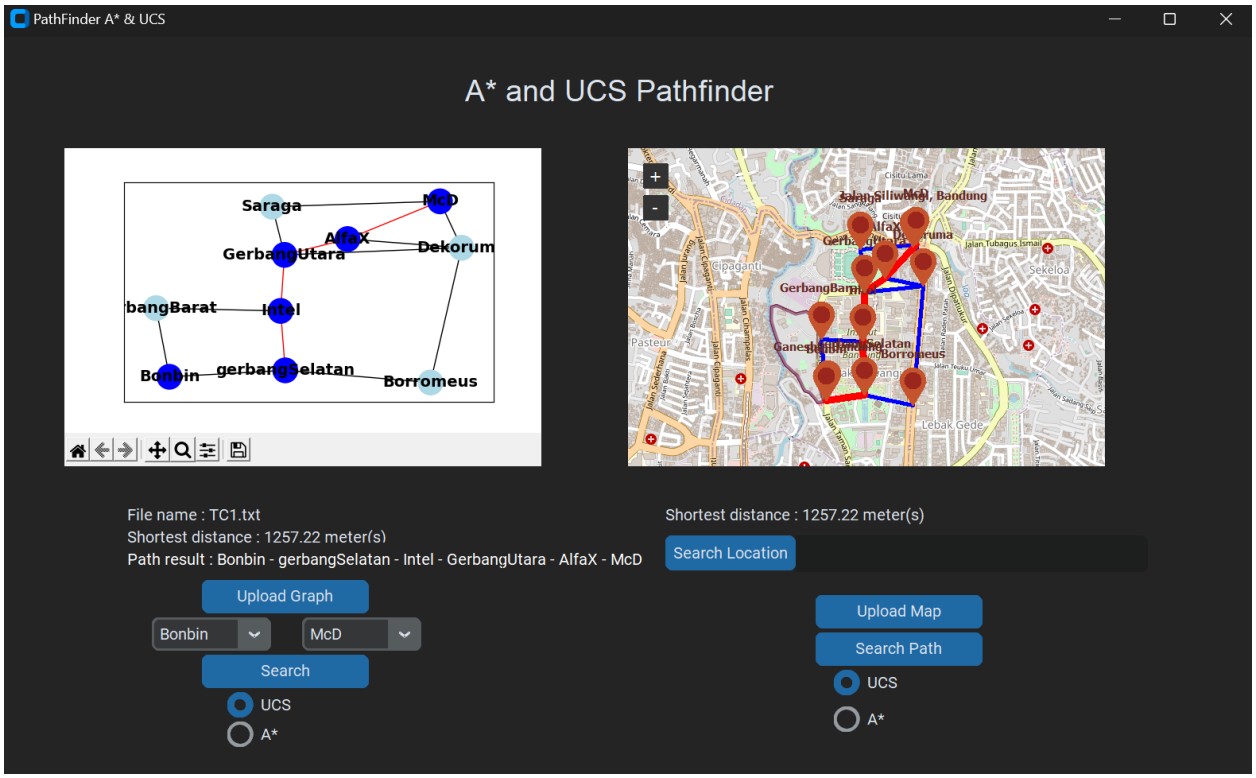
TC1.txt (Peta jalan sekitar kampus ITB) :

```
0 230.54 293.99 314.75 0 0 0 0 0 0
230.54 0 0 0 352.57 0 0 0 0 0
293.99 0 0 0 0 0 0 714.56 0 0
314.75 0 0 0 246.72 297.53 0 0 0 0
0 352.57 0 246.72 0 0 0 0 0 0
0 0 0 297.53 0 0 148.69 354.06 0 249.95
0 0 0 0 0 148.69 0 231.44 265.71 0
0 0 714.56 0 0 354.06 231.44 0 247.79 0
0 0 0 0 0 0 265.71 247.79 0 327.38
0 0 0 0 0 249.95 0 0 327.38 0
107.61045007292712 -6.893170180512400
107.60837897679768 -6.893477092920655
107.61304575144408 -6.893748287403765
107.61036744753650 -6.890341659009687
107.60814195889010 -6.890209869311779
107.61043665955694 -6.887674610904334
107.61156336496117 -6.886921698253013
107.61360103356867 -6.887335590694982
107.61321462738645 -6.885126403278361
107.61021883587576 -6.885381528287712
gerbangSelatan
Bonbin
Borromeus
Intel
GerbangBarat
GerbangUtara
AlfaX
Dekoruma
McD
Saraga
```

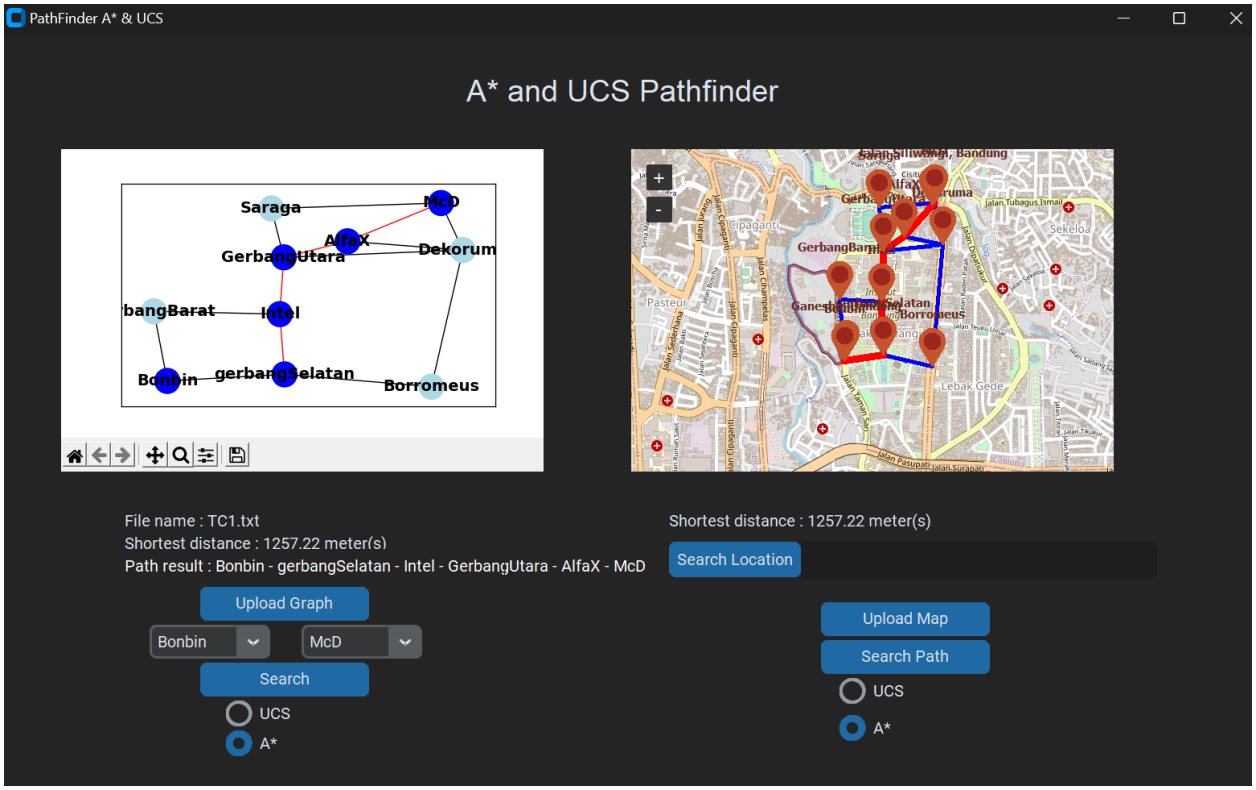
Isi file input



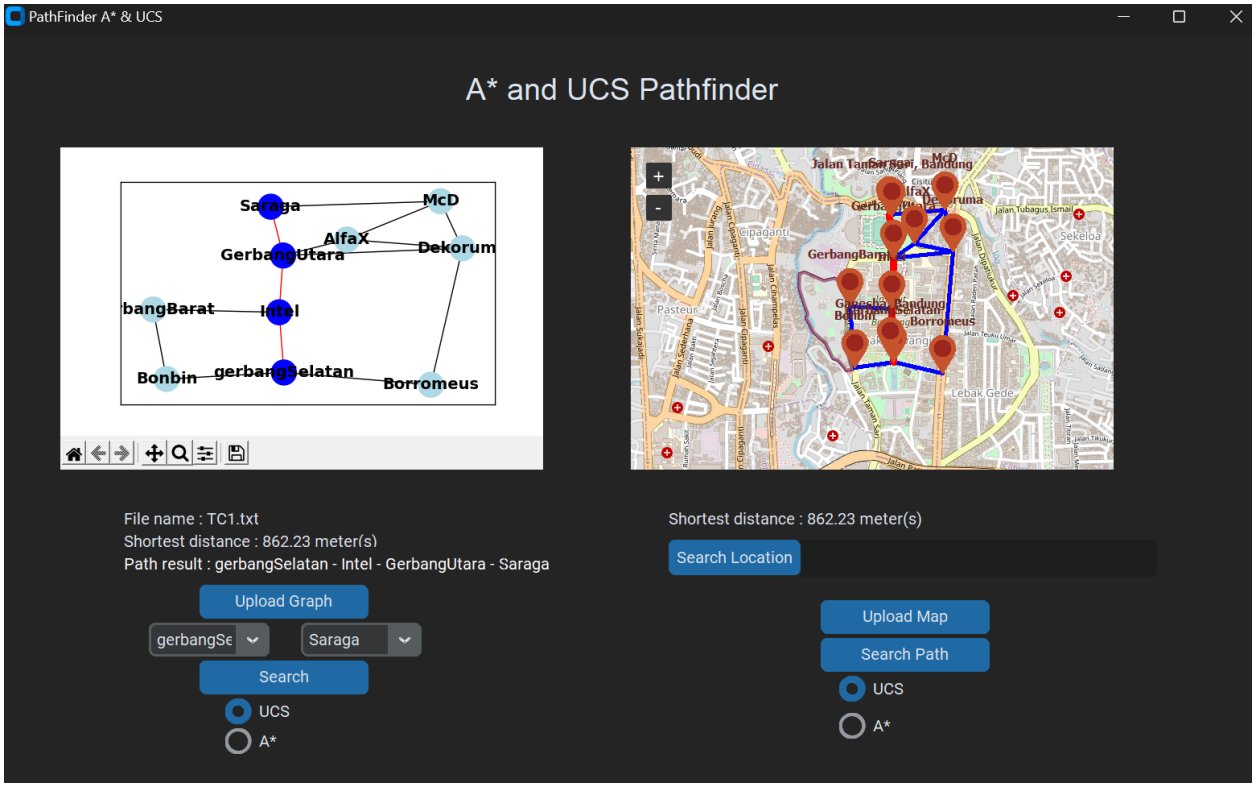
Visualisasi Graf



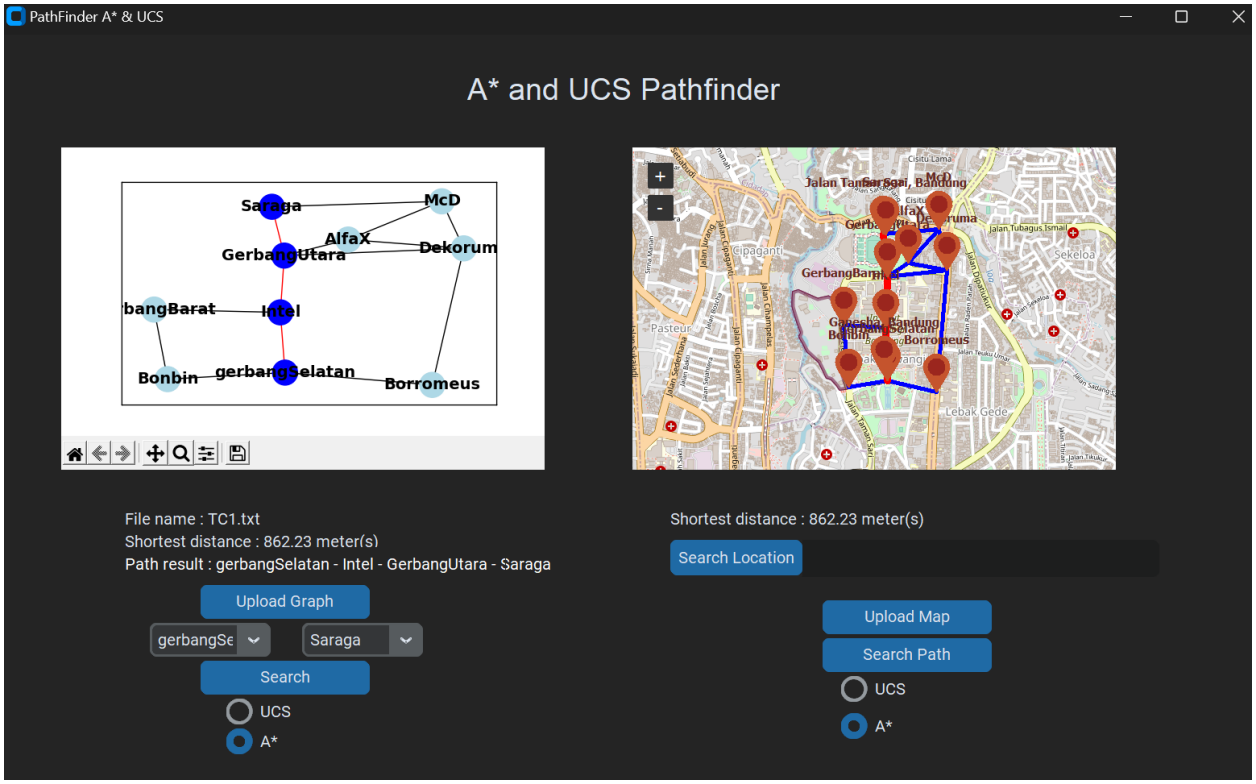
Jalur Bonbin -McD(UCS)



Jalur Bonbin-McD (A*)



Jalur gerbangSelatan - Saraga (UCS)

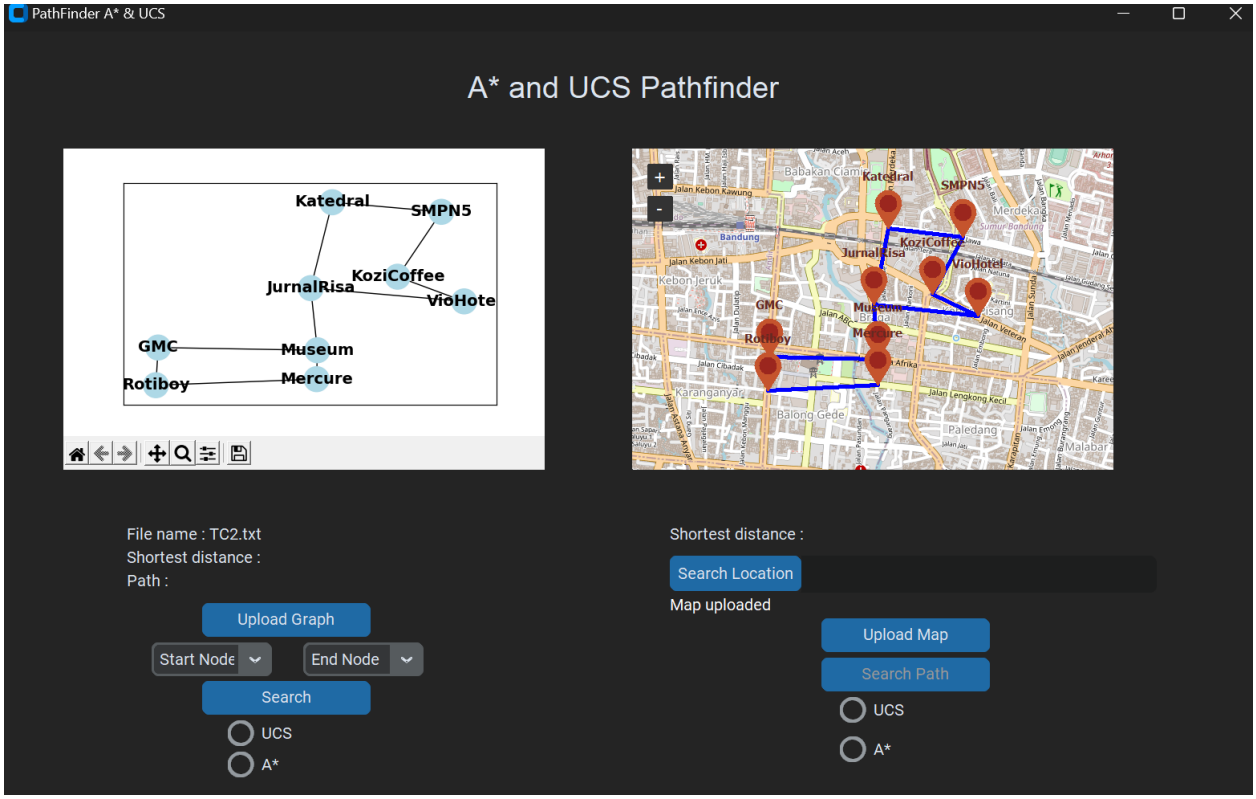


Jalur gerbangSelatan - Saraga (A*)

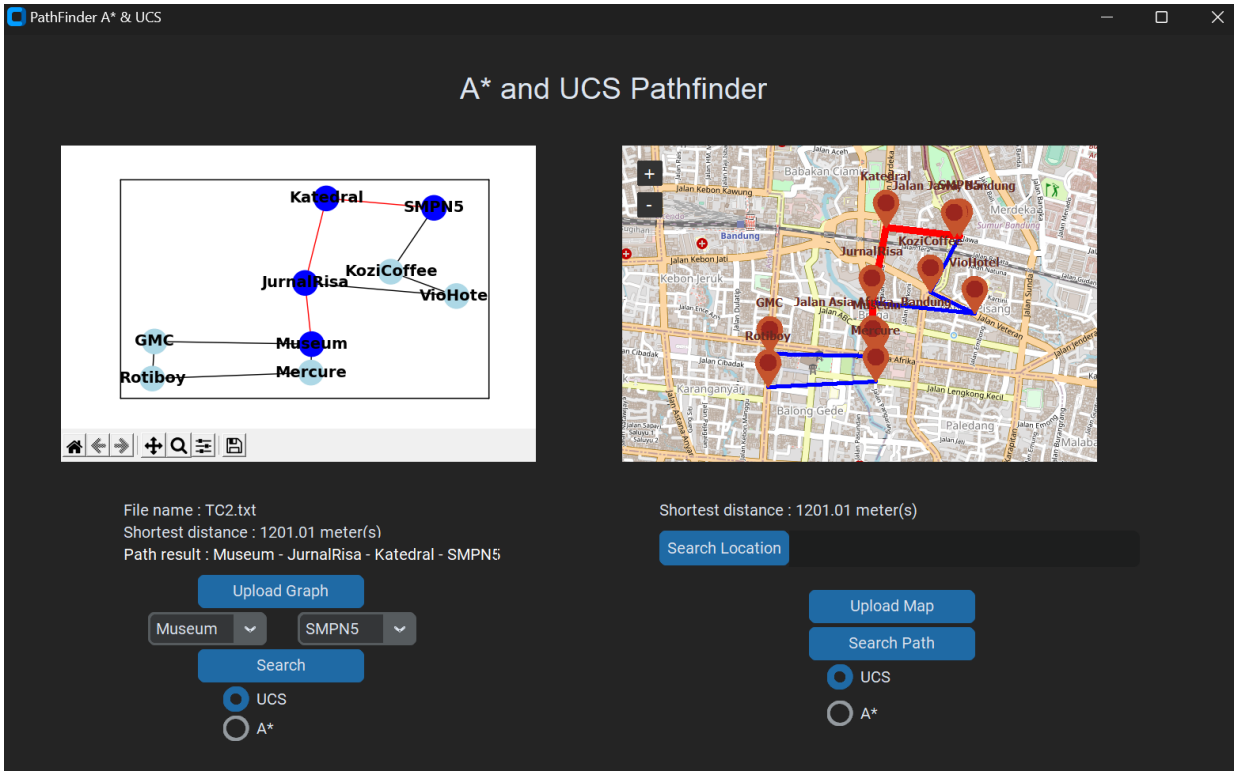
TC2.txt (Peta jalan sekitar Alun-Alun Bandung) :

```
0 318.02 0 0 0 0 607.21 0 157.43
318.02 0 621.07 461.39 0 0 0 0 0
0 621.07 0 0 0 219.93 0 0 0
0 461.39 0 0 421.60 0 0 0 0
0 0 0 421.60 0 369.87 0 0 0
0 0 219.93 0 369.87 0 0 0 0
607.21 0 0 0 0 0 0 215.79 0
0 0 0 0 0 0 215.79 0 650.49
157.43 0 0 0 0 0 0 650.49 0
107.60983099123675 -6.921464423709998
107.60958886367429 -6.918549055167118
107.61518089417696 -6.919171966369359
107.61037412381323 -6.914520097877944
107.61435150038865 -6.914969395457072
107.61275369031009 -6.918013738177005
107.60402761929028 -6.921336750459299
107.60394587600884 -6.923104827919552
107.60980353803714 -6.922820812345504
Museum
JurnalRisa
VioHotel
Katedral
SMPN5
KoziCoffee
GMC
Rotiboy
Mercure
```

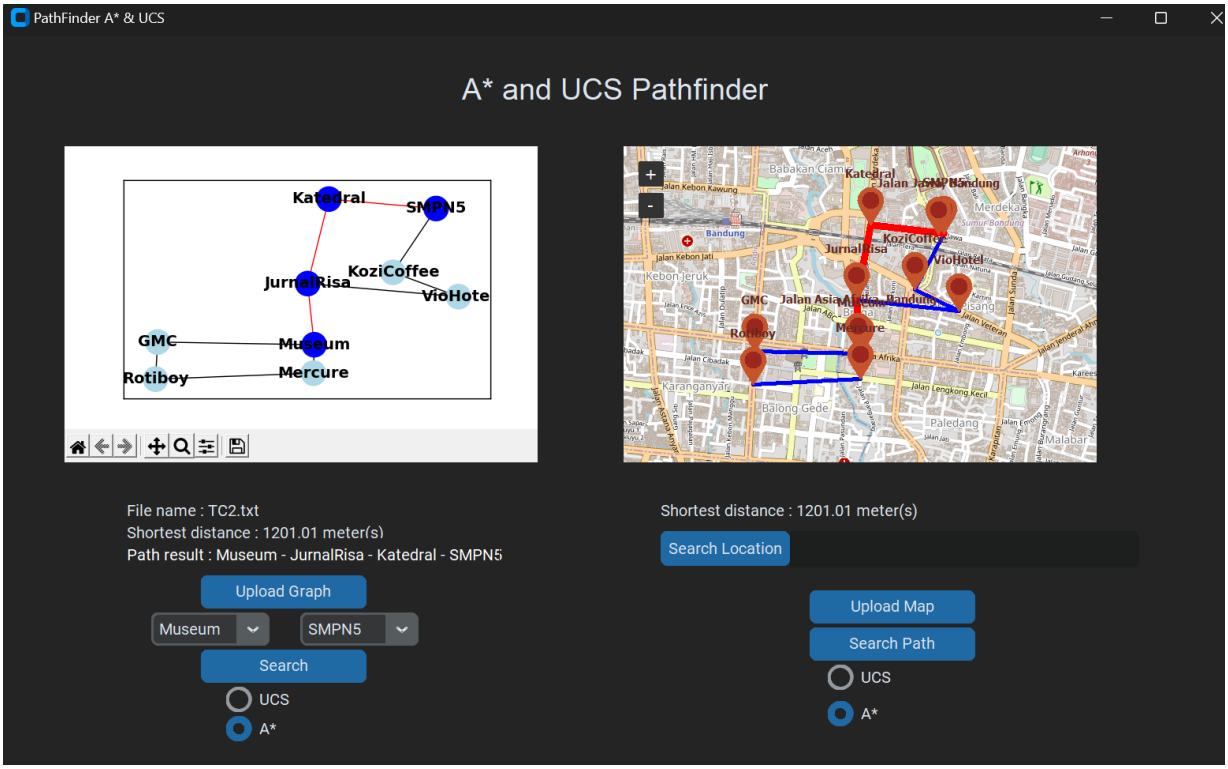
Isi file input



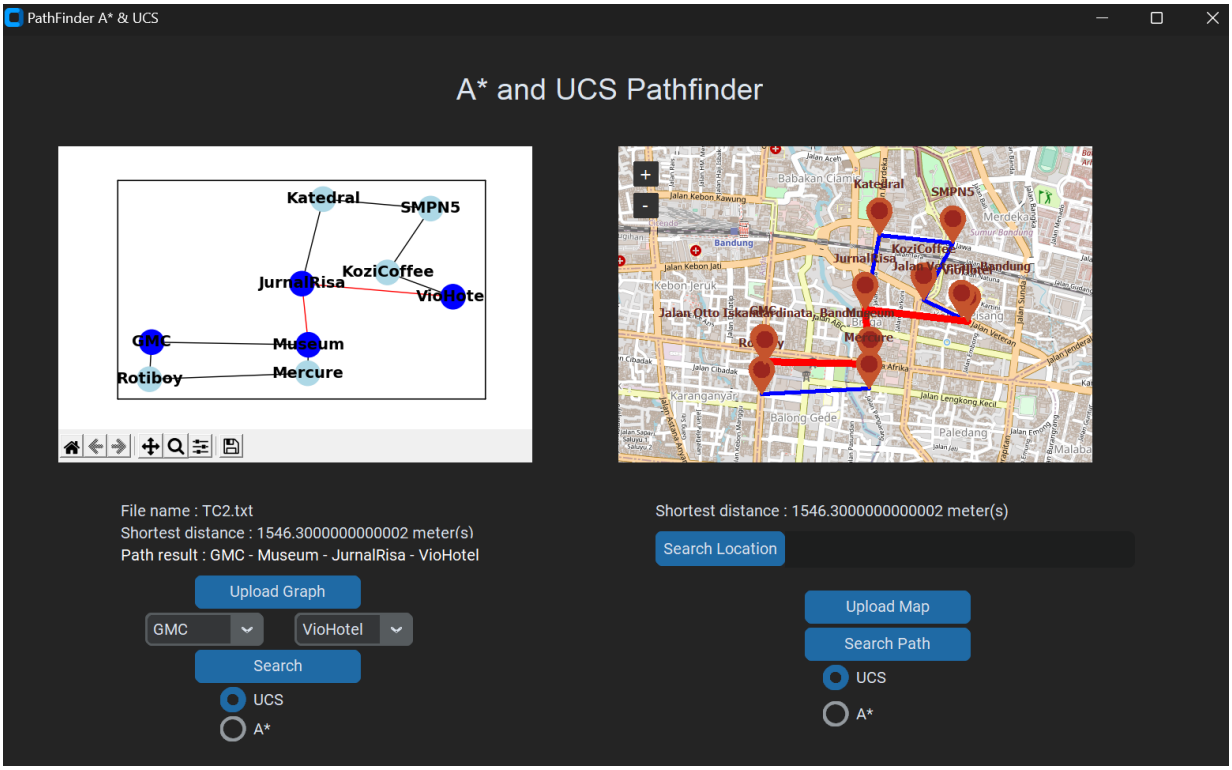
Visualisasi Graf



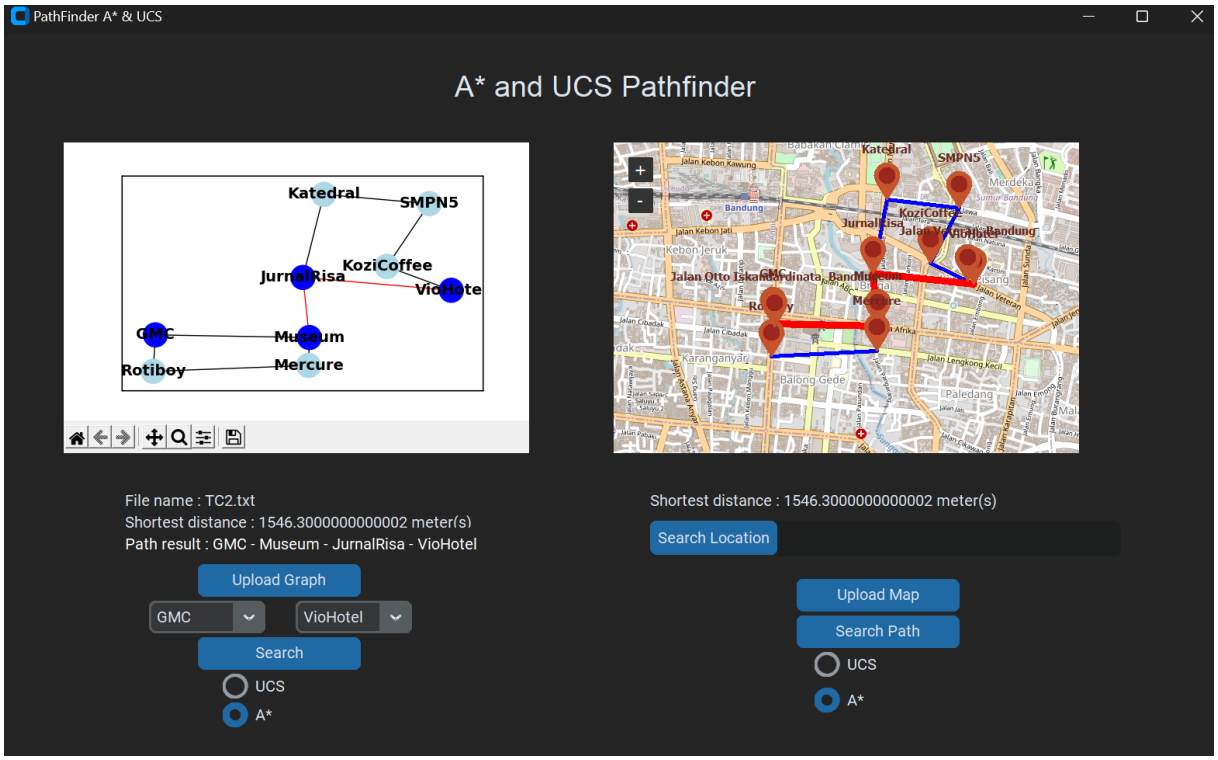
Jalur Museum - SMPN5 (UCS)



Jalur Museum - SMPN5 (A*)



Jalur GMC - VioHotel (UCS)

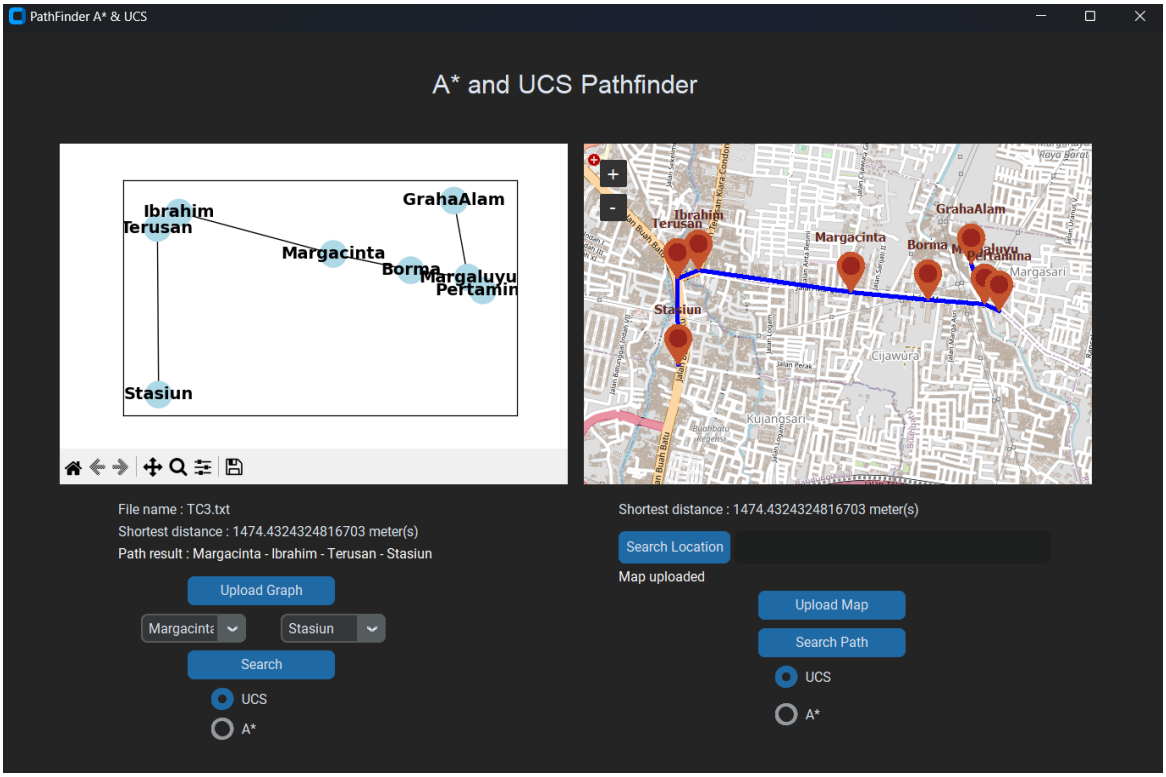


Jalur GMC - VioHotel (A*)

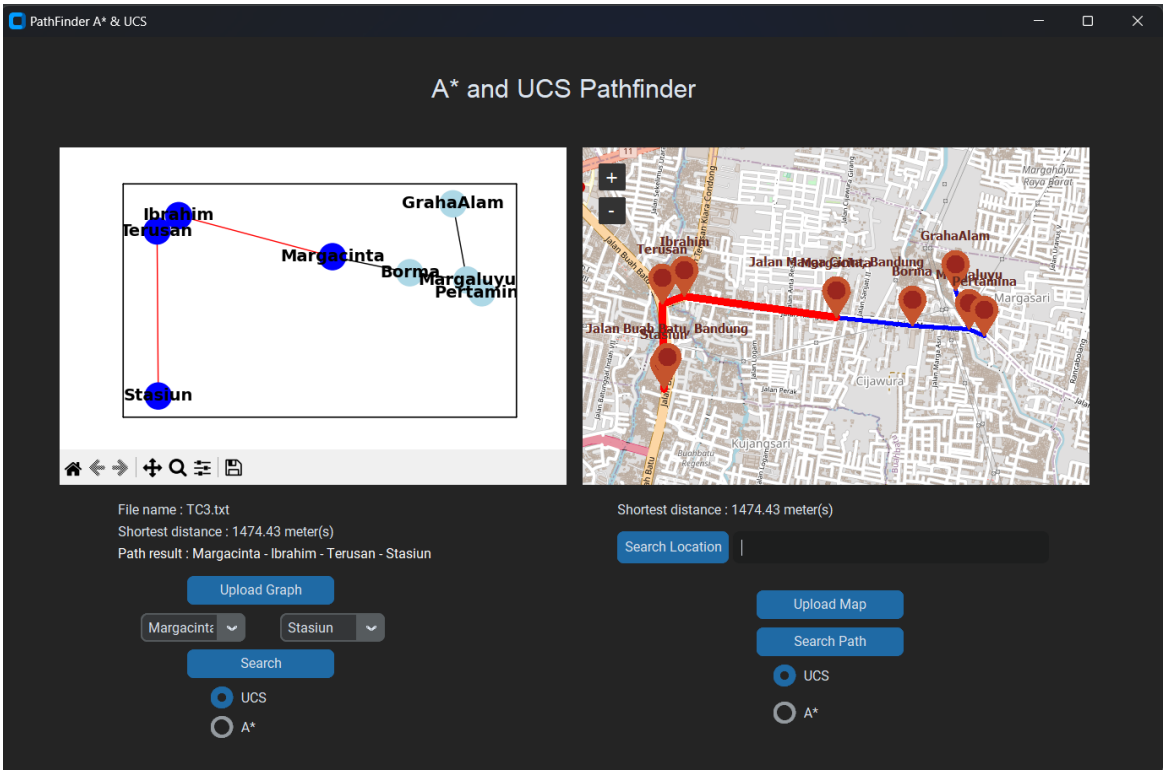
TC3.txt (Peta jalan sekitar Buahbatu) :

```
0 867.9490191193315 0 0 434.14260032539954 0 0 0
867.9490191193315 0 128.8747126370327 0 0 0 0 0
0 128.8747126370327 0 477.60870072530616 0 0 0 0
0 0 477.60870072530616 0 0 0 0 0
434.14260032539954 0 0 0 0 319.97147919783913 0 0
0 0 0 0 319.97147919783913 0 233.66565629197254 91.00453846818256
0 0 0 0 0 233.66565629197254 0 0
0 0 0 0 0 91.00453846818256 0 0
107.64793741557011 -6.95510360654422
107.6402012916933 -6.9540349895657085
107.63911248357955 -6.954433895343009
107.63917267228672 -6.958730873484742
107.65182125427465 -6.955521935872451
107.65469411980214 -6.955710094658227
107.65399369696213 -6.953727732606629
107.6554418326947 -6.9560438860739495
Margacinta
Ibrahim
Terusan
Stasiun
Borma
Margaluyu
GrahaAlam
Pertamina
```

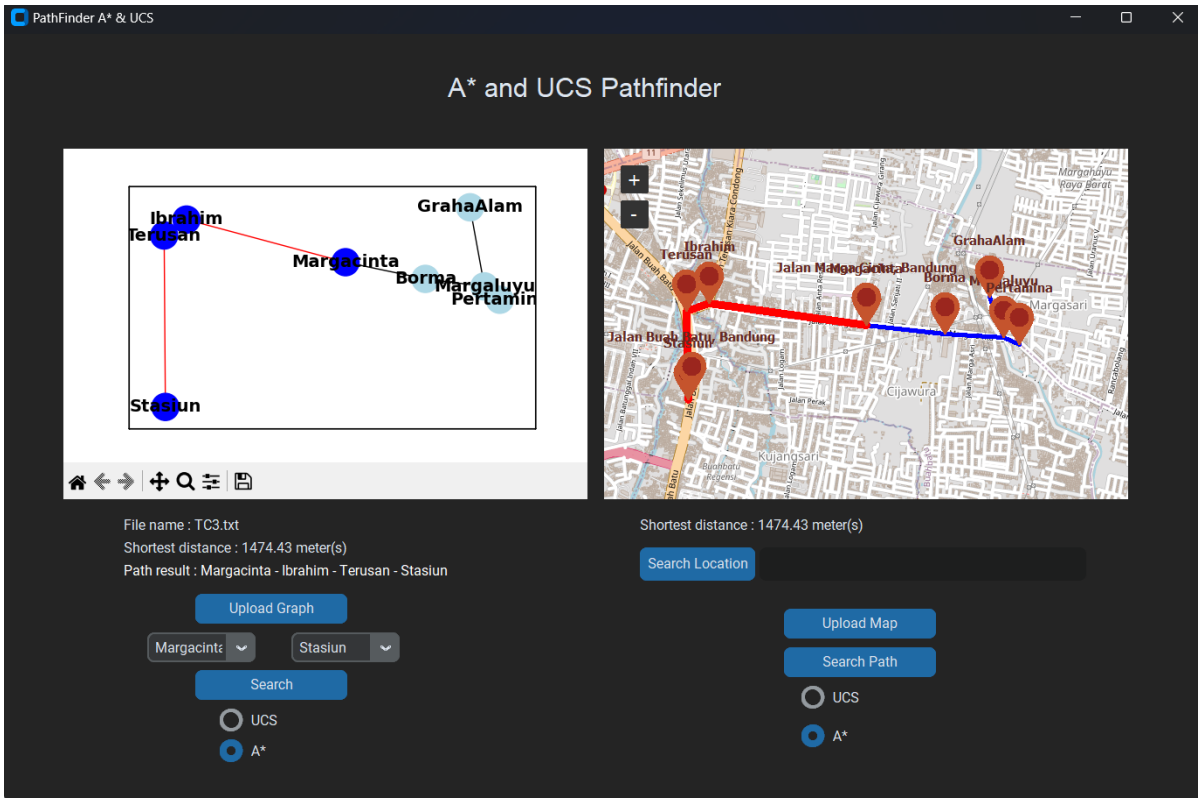
Isi file input



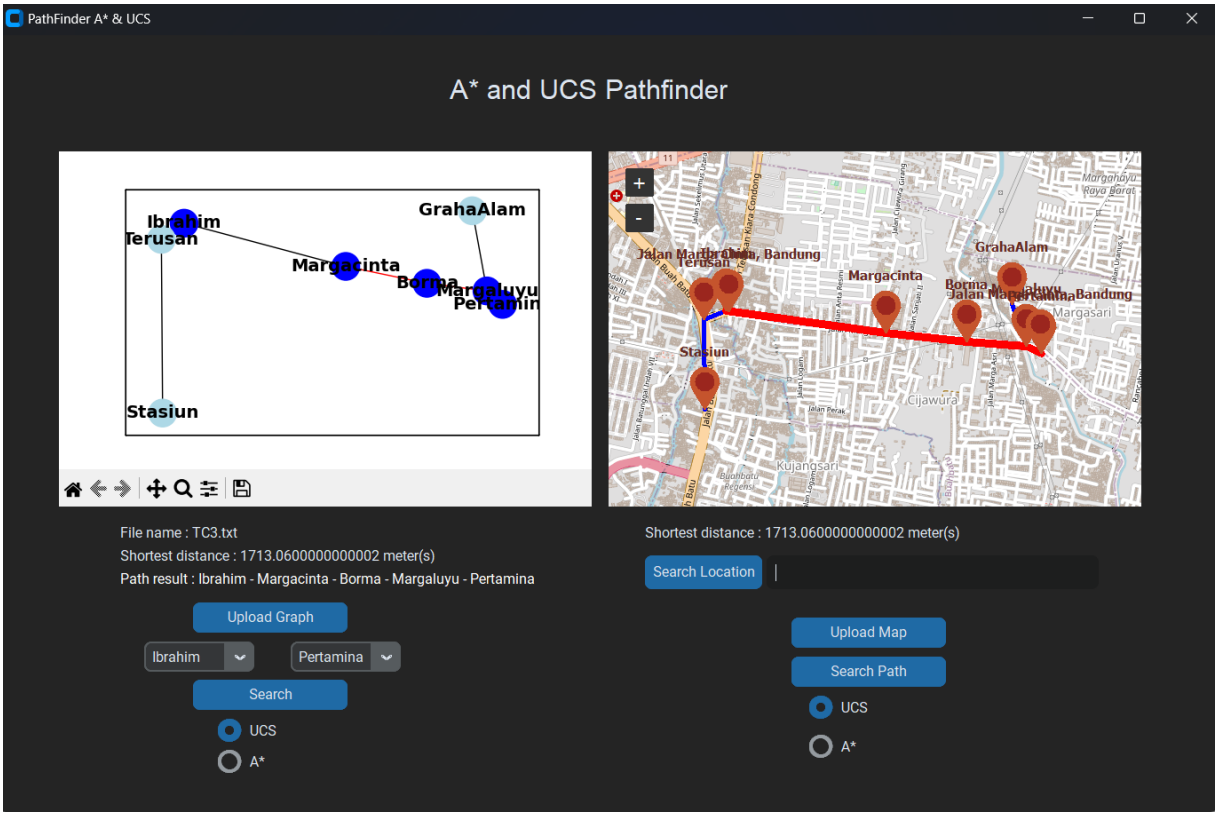
Visualisasi Graf



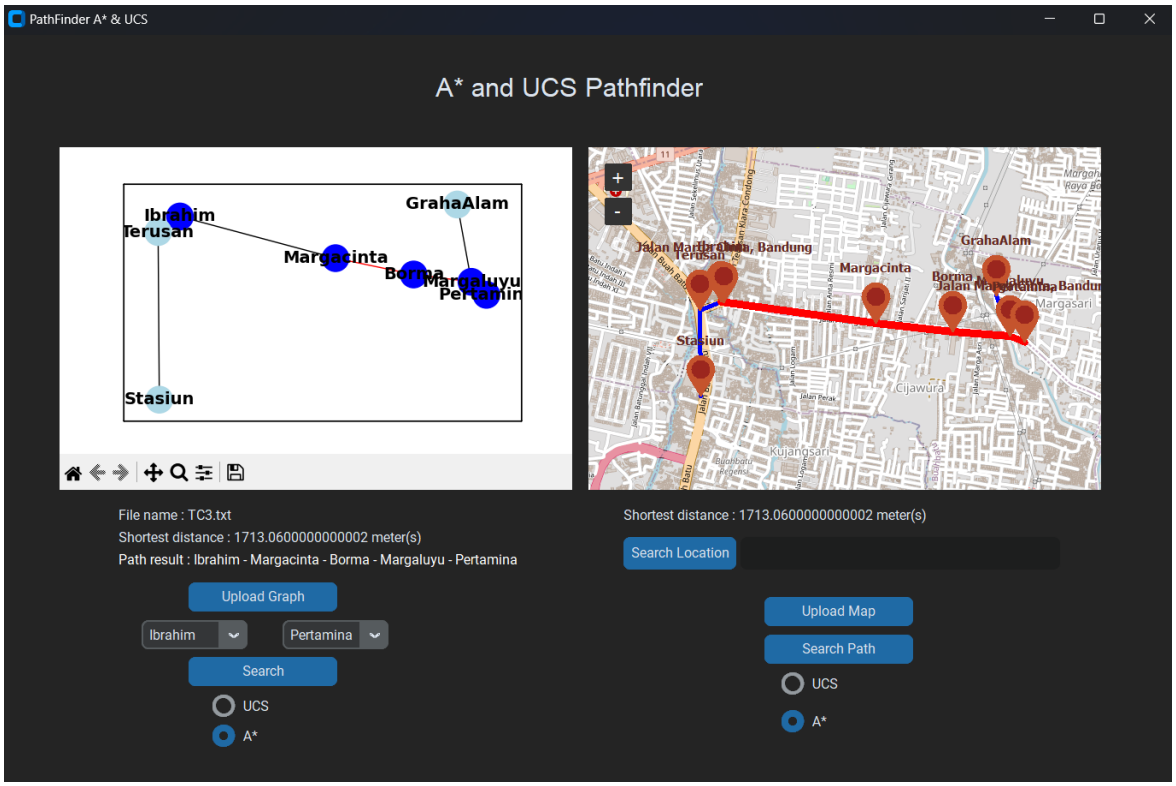
Jalur Margacinta-Statisun (UCS)



Jalur Margacinta-Statisun (A*)



Jalur Ibrahim-Pertamina (UCS)



Jalur Ibrahim-Pertamina (A*)

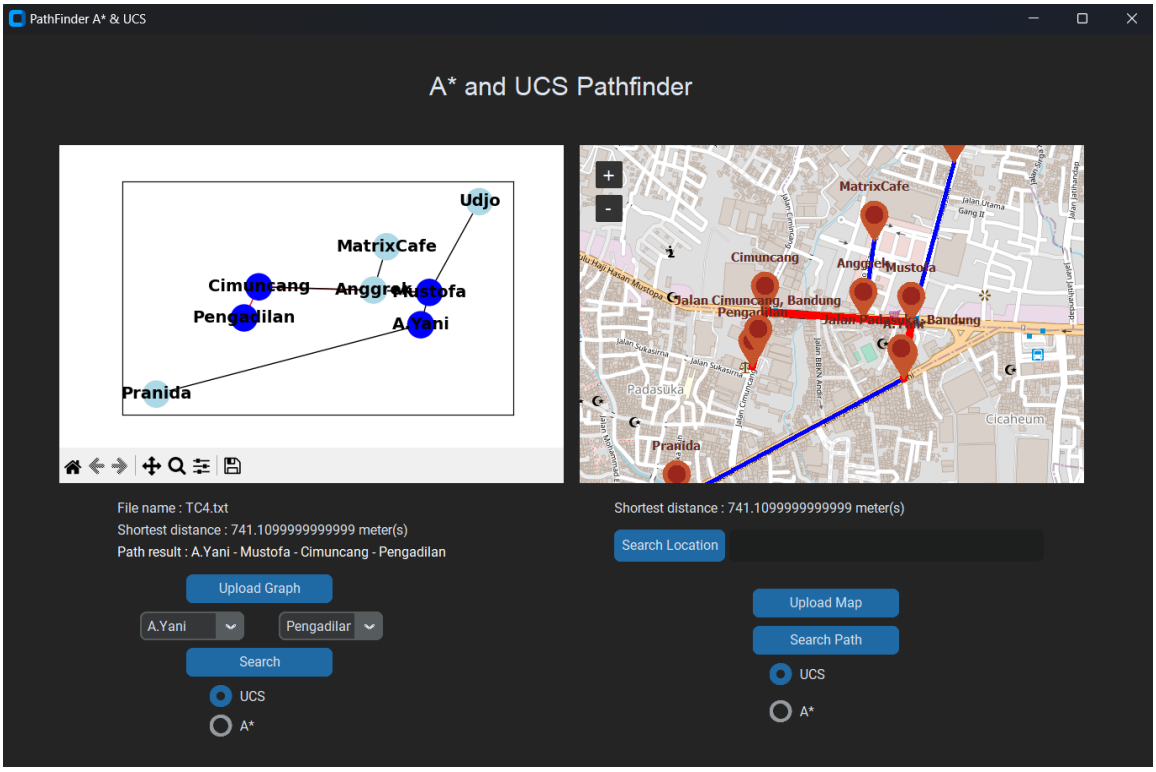
TC4.txt (Peta jalan kawasan Cibeunying Kidul di Kota Bandung) :

TC4

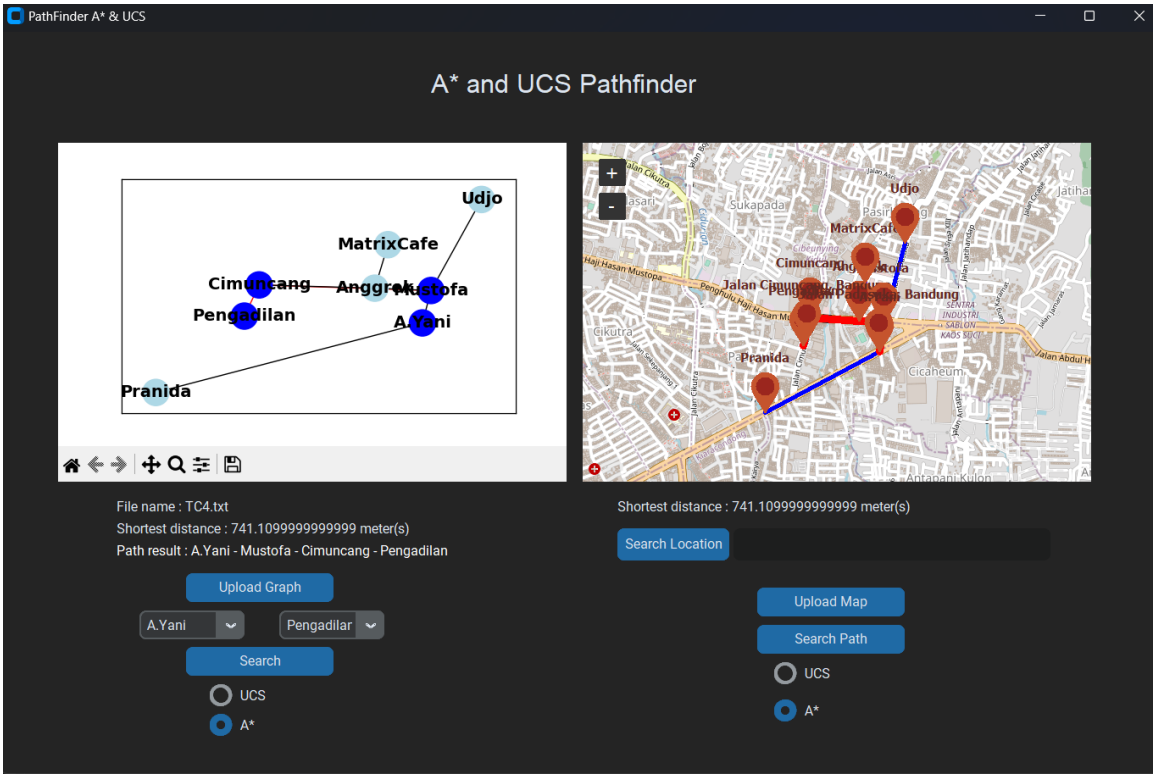
File Edit View

```
0 465.55 0 0 0 0 0 0
465.55 0 418.33 0 142.64 0 162.98 0
0 418.33 0 159.80 276.80 0 0 0
0 0 159.80 0 0 0 0 0
0 142.64 276.80 0 0 214.03 0 0
0 0 0 0 214.03 0 0 0
0 162.98 0 0 0 0 0 719.18
0 0 0 0 0 0 719.18 0
107.65457006729842 -6.8979357421527085
107.6534731880306 -6.901963444959681
107.64973788556888 -6.90171552043554
107.64941708996537 -6.903095002277445
107.65225201457416 -6.90185981798574
107.65253787592786 -6.899935819750884
107.65328588656244 -6.9033973077210575
107.64749289055078 -6.906474449829606
Udjo
Mustofa
Cimuncang
Pengadilan
Anggrek
MatrixCafe
A.Yani
Pranida
```

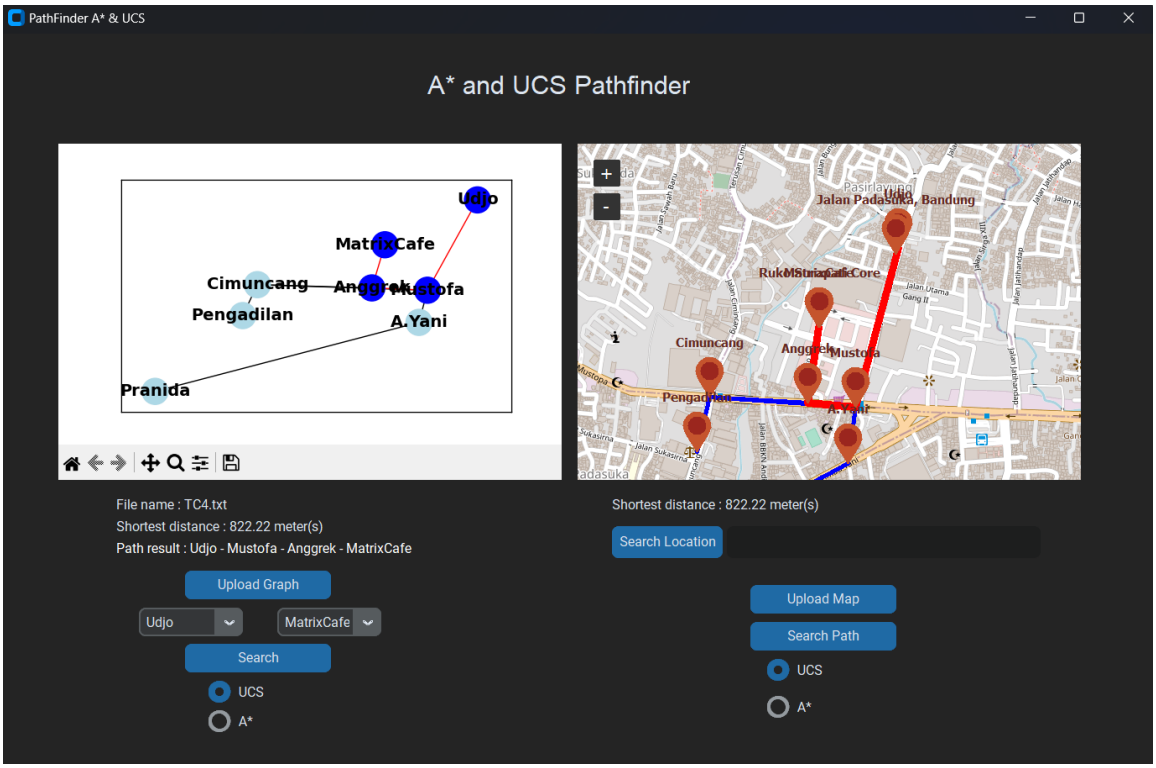
Isi file input



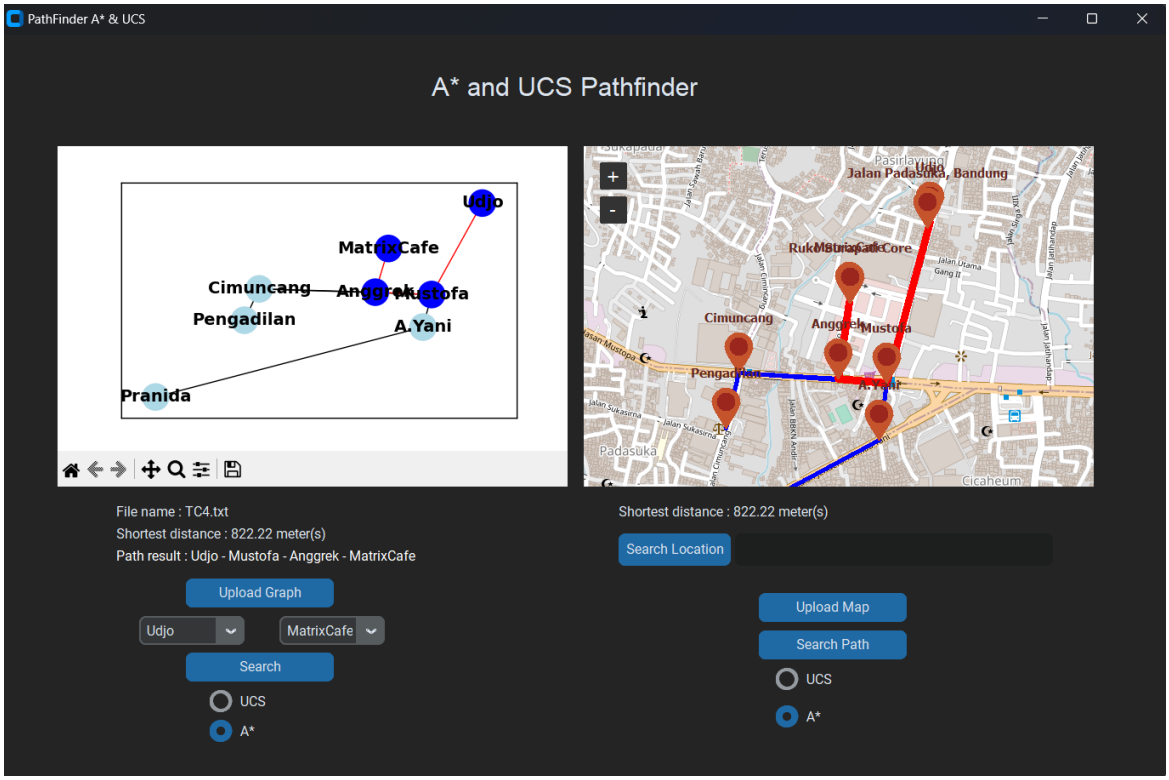
Jalur A.Yani - Pengadilan (UCS)



Jalur A.Yani - Pengadilan (A*)

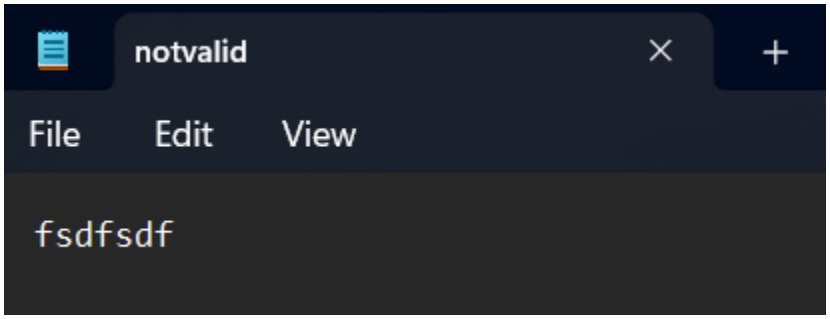


Jalur Udjo-MatrixCafe (UCS)

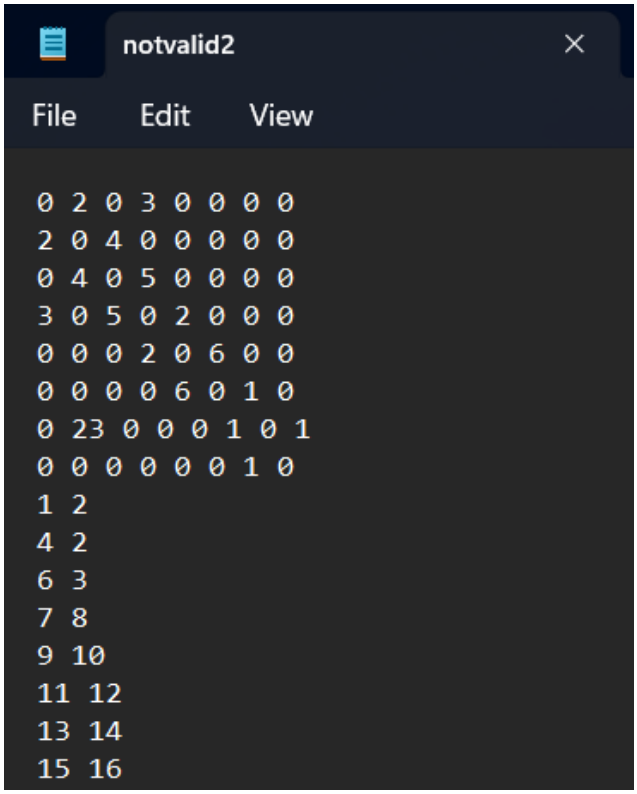


Jalur Udjo-MatrixCafe (A*)

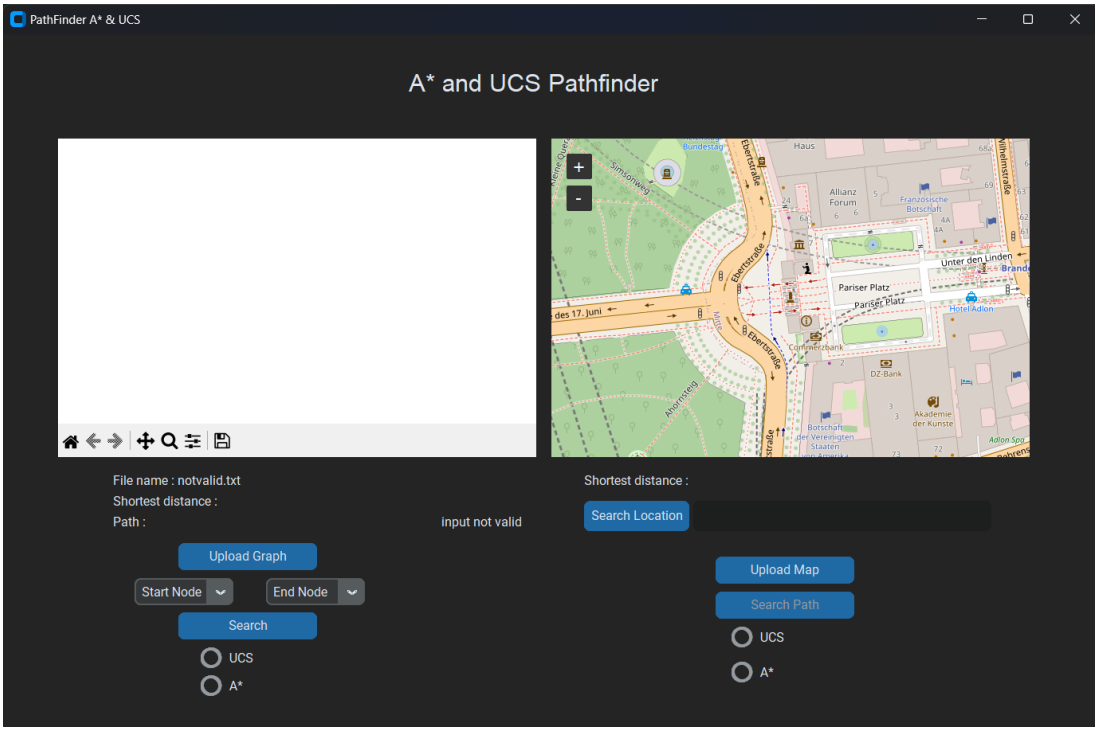
File tidak valid (notvalid.txt dan notvalid2.txt) :

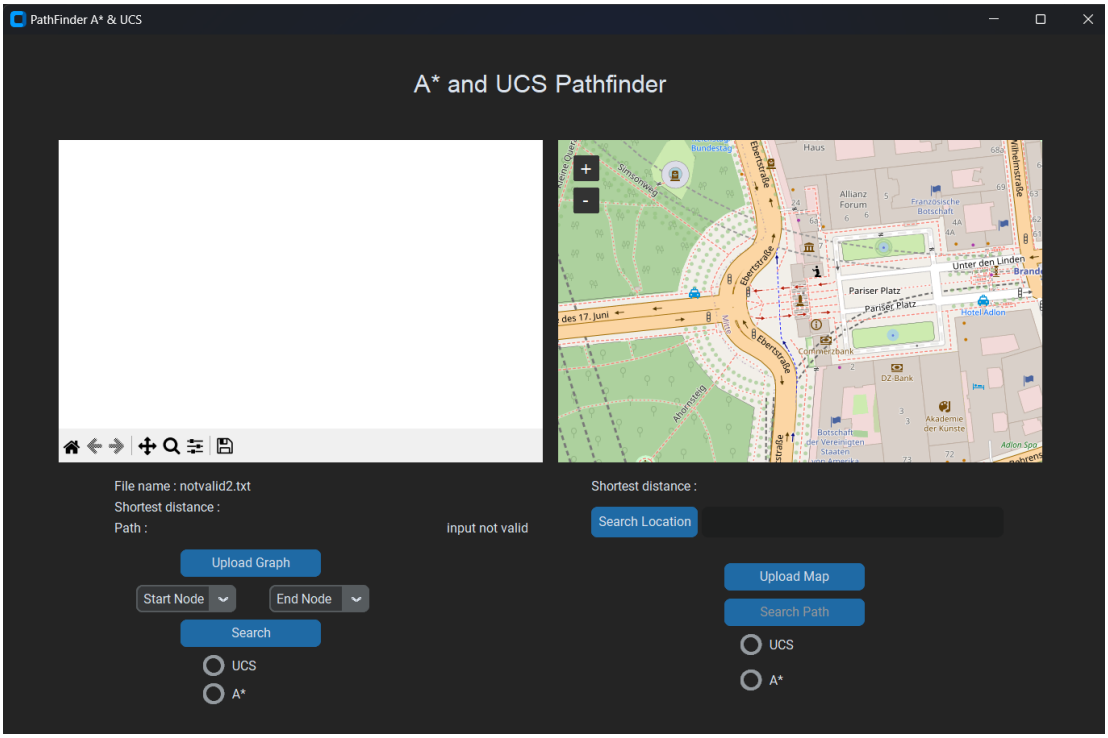


notvalid.txt



Notvalid2.txt (tidak ada nama simpul dan matriks tidak simetris)

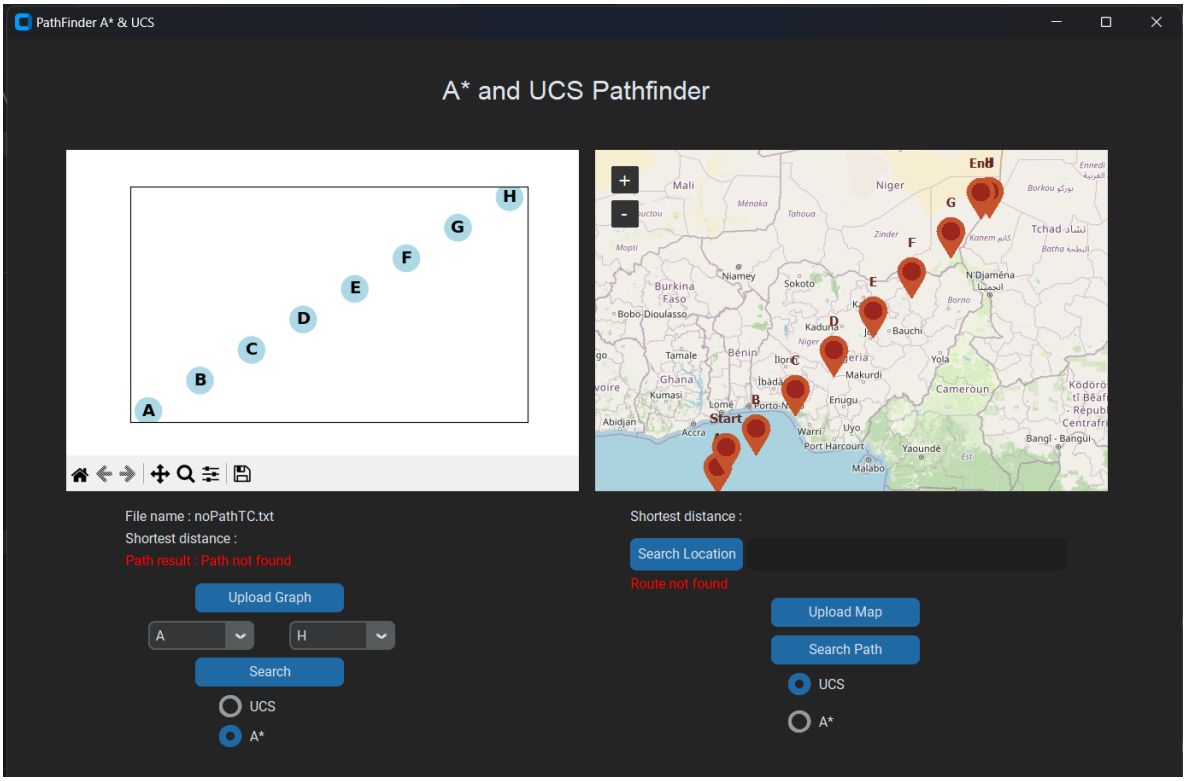




Jalur tidak ditemukan (noPathTC.txt) :

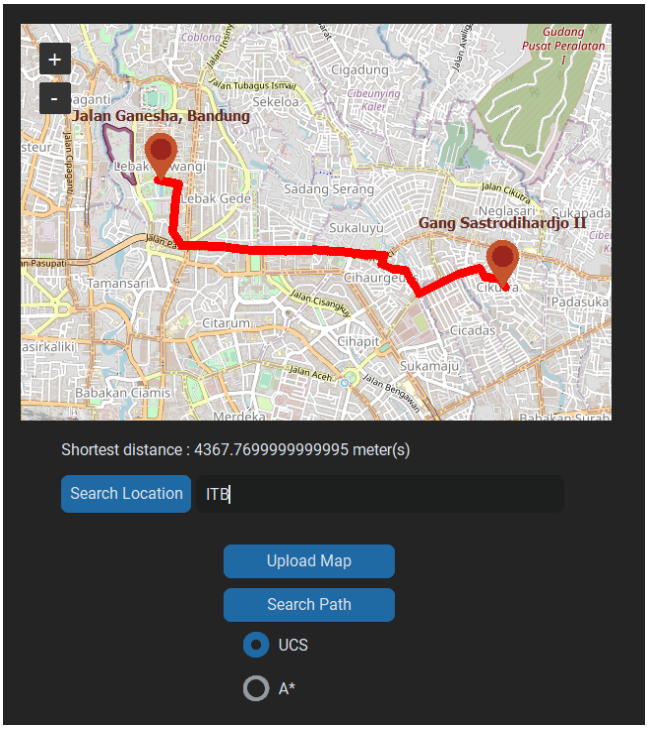
```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
A
B
C
D
E
F
G
H
```

Isi file input

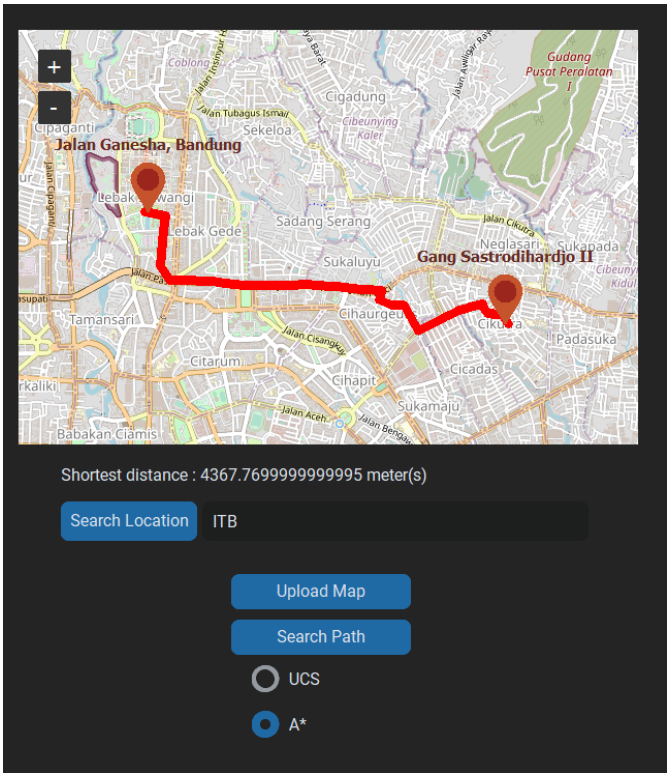


Jalur A-H (tidak ada)

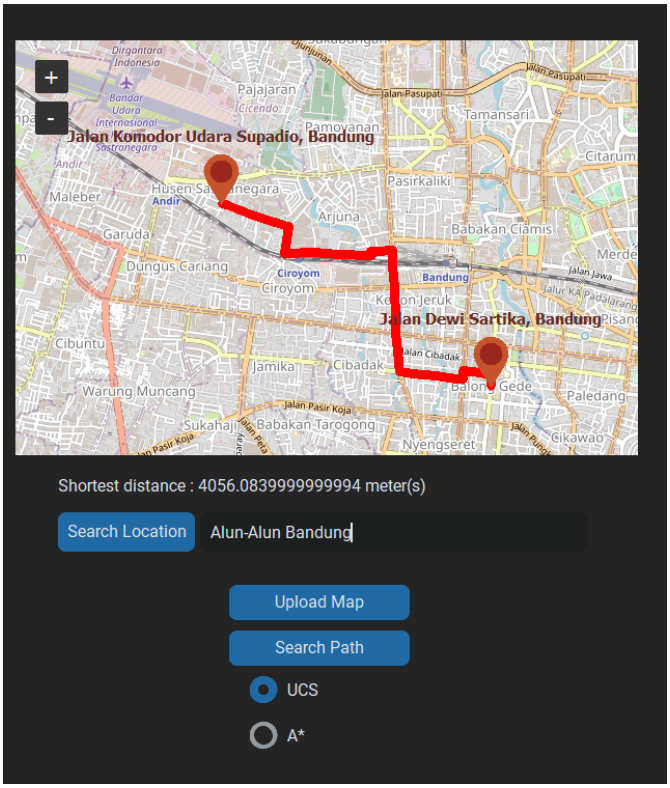
Jalur berdasarkan graf dari OSMNX :



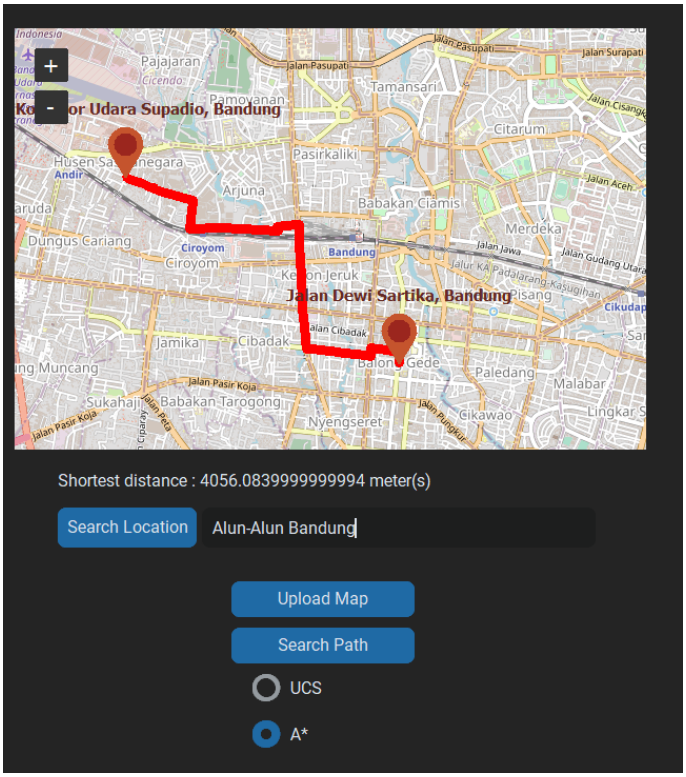
Jalur Gerbang Utama ITB - Cikutra (UCS)



Jalur Gerbang Utama ITB - Cikutra (A*)



Jalur Balong Gede - Jl. Komodor Udara Supadio (UCS)



Jalur Balong Gede - Jl. Komodor Udara Supadio (A*)

Bab IV

Kesimpulan dan Komentar

5.1 Kesimpulan

Penulis berhasil membuat program yang bisa mencari jalur terpendek pada peta ataupun graf masukan.

5.2 Komentar

Tugasnya sejauh ini paling rame dan paling keos, bahkan dibanding 2 tubes sebelumnya.

Lampiran

Tautan Repository Github : https://github.com/margarethaolivia/Tucil3_13521071_13521172

Bahasa yang digunakan : Python

Tabel Keberjalanan Program

Poin	Ya	Tidak
Program dapat menerima input graf	✓	
Program dapat menghitung lintasan terpendek dengan UCS	✓	
Program dapat menghitung lintasan terpendek dengan A*	✓	
Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓	