LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Oleh:

13518134 Muhammad Raihan Iqbal

13521068 Ilham Akbar

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

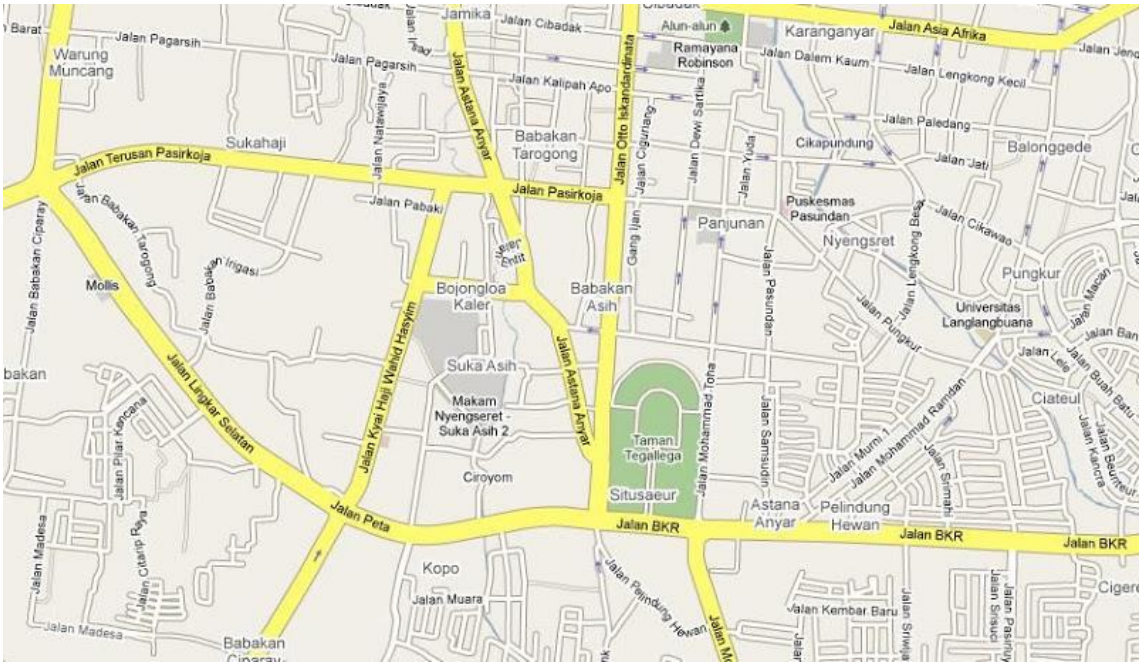
INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

BAB 1
DESKRIPSI PERSOALAN

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 antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.



Gambar 1. Peta sebagian wilayah Bandung
(sumber: Dokumen Spesifikasi Tugas Kecil 3 IF2211 Strategi Algoritma 2022/2023)

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.

Spesifikasi program:

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.

Bonus: Bonus nilai diberikan jika dapat menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta (berupa jalan yang diberi warna). Simpul graf diperoleh dari peta (menggunakan API Google Map) dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Euclidean.

BAB 2
IMPLEMENTASI PROGRAM

1. baca_file.py

```
import re

# membaca input dari file
def baca_file():
    # Membaca nama file dari user
    namaFile = input("Silakan masukkan nama file yang dipilih
(beserta extensi): ")
    print("")
    # Membaca isi file
    try:
        with open(namaFile, 'r') as file:
            # Membaca seluruh isi file
            file_contents = file.read()

            # Mengekstrak nilai variabel dari isi file menggunakan
ekspresi reguler
            banyakNode = int(re.search(r'jumalahNode\s*=\s*(\d+)',
file_contents).group(1))
            latitude = float(re.search(r'latitude\s*=\s*([\d.-]+)',
file_contents).group(1))
            longitude =
float(re.search(r'longitude\s*=\s*([\d.-]+)',
file_contents).group(1))
            listKoordinat =
eval(re.search(r'listKoordinat\s*=\s*(\[.*?\])', file_contents,
re.DOTALL).group(1))
            listNode = eval(re.search(r'listNode\s*=\s*(\[.*?\])',
file_contents, re.DOTALL).group(1))
            adjacencyMatrix =
eval(re.search(r'matrix\s*=\s*(\[.*?\])', file_contents,
re.DOTALL).group(1))

            # Mengembalikan nilai variabel dalam bentuk tuple
        return banyakNode, latitude, longitude, listKoordinat,
listNode, adjacencyMatrix

    # Jika file tidak ditemukan, maka akan memanggil fungsi
baca_file() kembali
    except:
        print("File tidak ditemukan, silakan coba lagi")
        baca_file()
```

2. graph.py

```
import Jarak_Euclidean as jarak
from queue import PriorityQueue

class Graph:
```

```
    # Inisiasi graf
    def __init__(self, banyakNode, listNode, listKoordinat,
adjacencyMatrix):
        self.__banyakNode = banyakNode
        self.__listnode = listNode
        self.__adjacencyMatrix = adjacencyMatrix
        self.__listKoordinat = listKoordinat
        self.__bobot = [[0 for y in range(self.__banyakNode)] for x
in range(self.__banyakNode)]

        # Mengubah adjacency matrix menjadi adjacency list
        self.__adjacencyList = []
        for x in range(self.__banyakNode):
            moves = []
            for y in range(self.__banyakNode):
                if(self.__adjacencyMatrix[x][y] == 1):
                    moves = moves + [y]
            self.__adjacencyList = self.__adjacencyList + [moves]

        # Menghitung bobot dari setiap edge
        for x in range(self.__banyakNode):
            for y in range(self.__banyakNode):
                if(self.__adjacencyMatrix[x][y] == 1):
                    m = self.__listKoordinat[x]
                    n = self.__listKoordinat[y]
                    bobot = jarak.euclidean(m,n)
                    self.__bobot[x][y] = bobot

    # mengembalikan path dari start ke tujuan
    def path(self, awal, curr):
        path = [curr]
        while curr in awal:
            curr = awal[curr]
            path += [curr]
        return path

    # mengecek apakah node ada di array_node
    def cek_node(self, node, array_node):
        for x in range(len(array_node)):
            if node == array_node[x]:
                return True
        return False

    # mengembalikan jumlah node
    def getNumNode(self):
        return self.__banyakNode

    # mengembalikan node
    def getNode(self, idxNode):
        return self.__listnode[idxNode]

    # mengembalikan index node
    def getIdxNode(self, node):
        for x in range(self.__banyakNode):
            if(self.__listnode[x] == node):
                return x
        return -1
```

```
    # mengembalikan list node dengan idx node yang dimasukkan
    def idxToNode(self, listnode):
        list = []
        for x in listnode:
            list += [self.getNode(x)]
        return list

    # mengembalikan list node
    def getListNode(self):
        return self.__listnode

    # mengembalikan adjacency matrix
    def getAdjacencyMatrix(self):
        return self.__adjacencyMatrix

    # mengembalikan list koordinat
    def getListKoordinat(self):
        return self.__listKoordinat

    # mengembalikan koordinat node
    def getNodeKoordinat(self, idxNode):
        return self.__listKoordinat[idxNode]

    # mengembalikan koordinat X node
    def getKoordinatX(self, idxNode):
        return self.getNodeKoordinat(idxNode)[0]

    # mengembalikan koordinat Y node
    def getKoordinatY(self, idxNode):
        return self.getNodeKoordinat(idxNode)[1]

    # mengembalikan adjacency list
    def getAdjacencyList(self):
        return self.__adjacencyList

    # mengembalikan bobot
    def getBobot(self):
        return self.__bobot

    # mengembalikan adjacency list
    def getAdjacencyList(self):
        return self.__adjacencyList

    # mengembalikan list koordinat X
    def idxToKoordinatX(self, listnode):
        koordinat_X = []
        for x in listnode:
            koordinat_X = koordinat_X +
[self.getNodeKoordinat(x)[0]]
        return koordinat_X

    # mengembalikan list koordinat Y
    def idxToKoordinatY(self, listnode):
        koordinat_Y = []
        for x in listnode:
            koordinat_Y = koordinat_Y +
[self.getNodeKoordinat(x)[1]]
        return koordinat_Y
```

```
    # mengembalikan moves dari node
    def getmoves(self, kemana):
        return self.__adjacencyList[kemana]

    # # mengembalikan bobot dari node
    # def getBobot(self, dari, ke):
    #     return self.__bobot[dari][ke]

    # algoritma A*
    def a_star(self, start, tujuan):
        # inisiasi
        banyakNode = self.__banyakNode
        count = 0
        antrian = PriorityQueue()
        antrian.put((0, count, start))
        asal = {}

        # inisiasi g(n) dan f(n)
        g = [float("inf") for x in range(banyakNode)]
        g[start] = 0

        f = [float("inf") for x in range(banyakNode)]
        f[start] =
jarak.euclidean(self.__listKoordinat[start],self.__listKoordinat[tuj
uan])

        # inisiasi dikunjungi
        dikunjungi = {start}

        # looping
        while (not antrian.empty()):
            # mengambil node dengan f(n) terkecil
            current = antrian.get()[2]
            dikunjungi.remove(current)

            # jika node tujuan ditemukan
            if(current == tujuan):
                return self.path(asal, current)

            # looping untuk setiap node yang bisa diakses dari node
current
            for moves in self.__adjacencyList[current]:
                # menghitung g(n) dan f(n)
                temp_g = g[current] +
jarak.euclidean(self.__listKoordinat[current],
self.__listKoordinat[moves])
                # jika g(n) lebih kecil dari g(n) sebelumnya
                if(temp_g < g[moves]):
                    asal[moves] = current
                    g[moves] = temp_g
                    f[moves] = temp_g +
jarak.euclidean(self.__listKoordinat[moves],
self.__listKoordinat[current])
                    # jika node moves belum ada di antrian
                    if (moves not in dikunjungi):
                        count = count + 1
                        antrian.put((f[moves], count, moves))
```

```
                        dikunjungi.add(moves)

        return None
```

3. Jarak_Euclidean.py

```
import math

def euclidean(x,y):

    # latitude dan longitude
    latitude_x = x[0]
    longitude_x = x[1]
    latitude_y = y[0]
    longitude_y = y[1]

     # convert ke radian
    x_rad = latitude_x * math.pi / 180.0
    y_rad = latitude_y * math.pi / 180.0

    # selisih
    selisih_latitude = (latitude_y - latitude_x) * math.pi / 180.0
# 1 derajat = 1/180 * pi rad
    selisih_longitude = (longitude_y - longitude_x) * math.pi /
180.0

    # akar
    a = (pow(math.sin(selisih_latitude / 2), 2) +
pow(math.sin(selisih_longitude / 2), 2) * math.cos(x_rad) *
math.cos(y_rad))

    # jari jari bumi
    r = 6371

    # rumus untuk mendapatkan distance dalam meter
    hasil = 2*r*math.asin(math.sqrt(a))*1000

    return hasil
```

4. Main.py

```
import graph as g
import Jarak_Euclidean as jarak
import gmplot
import baca_file as baca
import UCS as ucs

def main():
    # Menampilkan judul program
    print("Program Jarak Terpendek dengan A* dan UCS")
    print("----------------------------------------")
    # Memanggil fungsi baca_file() untuk membaca input dari file
    numNode, latitude, longitude, listKoordinat, listNode,
adjacencyMatrix = baca.baca_file()
```

```
    # Menampilkan nilai variabel yang telah dibaca dari file
    print("Banyak Node :", numNode)
    print("")
    print("Latitude :", latitude)
    print("")
    print("Longitude :", longitude)
    print("")
    print("Koordinat :")
    for koordinat in listKoordinat:
        print(koordinat)
    print("")
    print("Adjacency Matrix:")
    for row in adjacencyMatrix:
        print(row)
    print("")
    print("Nama Jalan : ")
    for node in listNode:
        print('>',node)
    print("")

    # Inisiasi graf
    graf = g.Graph(numNode, listNode, listKoordinat,
adjacencyMatrix)

    # Meminta Input Pengguna
    start = input("Masukkan Start (Nama Jalan) : ")
    while not graf.cek_node(start, listNode):
        print("Node Tidak ditemukan, silakan input ulang")
        start = input("Masukkan Start (Nama Jalan) : ")
    tujuan = input("Masukkan Tujuan (nama jalan) : ")
    while not graf.cek_node(tujuan, listNode):
        print("Node Tidak ditemukan, silakan input ulang")
        tujuan = input("Masukkan Tujuan (nama jalan) : ")

    # Memilih algoritma yang digunakan
    print("1. A*")
    print("2. UCS")
    pilihan = input("Pilih algoritma yang digunakan (1 atau 2) : ")

    if pilihan == "1":
        # Menjalankan A*
        list_arah = graf.a_star(graf.getIdxNode(start),
graf.getIdxNode(tujuan))
        # Menghitung bobot
        bobot = 0
        for i in range(len(list_arah)-1):
            bobot += jarak.euclidean(listKoordinat[list_arah[i]],
listKoordinat[list_arah[i+1]])

        # inisiasi node yang dikunjungi
        node_dikunjungi = graf.idxToNode(list_arah)
        node_dikunjungi_reversed = node_dikunjungi[::-1]
        print()
        print("Rute Terpendeknya adalah : ")
        # Menampilkan rute terpendek
        kunjungan = 0
        for node in node_dikunjungi_reversed:
```

```
            if kunjungan == 0:
                print(node, end='')
            elif kunjungan == numNode:
                print(" -> ",node)
            else:
                print(" -> ",node,end='')
            kunjungan += 1

        print()
        print("Jarak terpendek dari " + start + " menuju " + tujuan
+ " adalah " + str(bobot) + " meter.")
        print()
        ## Plotter

        rute_latitude = graf.idxToKoordinatX(list_arah)
        rute_longitude = graf.idxToKoordinatY(list_arah)

        map_latitude = latitude
        map_longitude = longitude

        gmap = gmplot.GoogleMapPlotter(map_latitude, map_longitude,
18)
        for i in range(numNode):
            for j in range(numNode):
                if(adjacencyMatrix[i][j] == 1):
                    latitude = [graf.getKoordinatX(i),
graf.getKoordinatX(j)]
                    longitude = [graf.getKoordinatY(i),
graf.getKoordinatY(j)]
                    gmap.scatter(latitude, longitude, 'blue', size =
4.5, marker=False)
                    gmap.plot(latitude, longitude, 'black',
edge_width = 2.5)

        gmap.scatter(rute_latitude, rute_longitude, 'red', size =
4.5, marker=False)
        gmap.plot(rute_latitude, rute_longitude, 'red', edge_width =
2.5)

        gmap.marker(rute_latitude[0], rute_longitude[0], 'red',
label='S', title='titik start', info_window="<text>Titik
start</text>")
        gmap.marker(rute_latitude[-1], rute_longitude[-1], 'green',
label='F', title='titik finish', info_window="<text>Titik
finish</text>")

        # Menyimpan peta dalam file HTML
        gmap.draw("output-astar.html")
        print("")
        print("Peta telah disimpan dalam file output-astar.html di
folder yang sama dengan program ini.")

    elif pilihan == "2":
        list_arah = ucs.ucs(graf, graf.getIdxNode(start),
graf.getIdxNode(tujuan))
        # Menghitung bobot
        bobot = 0
        for i in range(len(list_arah)-1):
```

```
            bobot += jarak.euclidean(listKoordinat[list_arah[i]],
listKoordinat[list_arah[i+1]])

        # inisiasi node yang dikunjungi
        node_dikunjungi = graf.idxToNode(list_arah)
        node_dikunjungi_reversed = node_dikunjungi[::-1]
        print()
        print("Rute Terpendeknya adalah : ")
        # Menampilkan rute terpendek
        kunjungan = 0
        for node in node_dikunjungi_reversed:
            if kunjungan == 0:
                print(node, end='')
            elif kunjungan == numNode:
                print(" -> ",node)
            else:
                print(" -> ",node,end='')
            kunjungan += 1

        print()
        print("Jarak terpendek dari " + start + " menuju " + tujuan
+ " adalah " + str(bobot) + " meter.")
        print()
        ## Plotter

        rute_latitude = graf.idxToKoordinatX(list_arah)
        rute_longitude = graf.idxToKoordinatY(list_arah)

        map_latitude = latitude
        map_longitude = longitude

        gmap = gmplot.GoogleMapPlotter(map_latitude, map_longitude,
18)
        for i in range(numNode):
            for j in range(numNode):
                if(adjacencyMatrix[i][j] == 1):
                    latitude = [graf.getKoordinatX(i),
graf.getKoordinatX(j)]
                    longitude = [graf.getKoordinatY(i),
graf.getKoordinatY(j)]
                    gmap.scatter(latitude, longitude, 'blue', size =
4.5, marker=False)
                    gmap.plot(latitude, longitude, 'black',
edge_width = 2.5)

        gmap.scatter(rute_latitude, rute_longitude, 'red', size =
4.5, marker=False)
        gmap.plot(rute_latitude, rute_longitude, 'red', edge_width =
2.5)

        gmap.marker(rute_latitude[0], rute_longitude[0], 'red',
label='S', title='titik start', info_window="<text>Titik
start</text>")
        gmap.marker(rute_latitude[-1], rute_longitude[-1], 'green',
label='F', title='titik finish', info_window="<text>Titik
finish</text>")

        gmap.draw("output-ucs.html")
```

```
        print("")
        print("Peta telah disimpan dalam file output-ucs.html di
folder yang sama dengan program ini.")

    else:
        print("Pilihan algoritma tidak valid. Program dihentikan.")


if __name__ == "__main__":
    main()
```

5. UCS.py

```
import heapq
import graph as graf
import Jarak_Euclidean as jarak

# Implementasi algoritma Uniform Cost Search (UCS)
def ucs(graf, start, goal):
    # Inisialisasi heap (prioritas antrian)
    heap = []
    # Memasukkan titik awal ke dalam heap
    heapq.heappush(heap, (0, start))

    # Inisialisasi set untuk menyimpan node yang sudah dieksplorasi
    explored = set()

    # Inisialisasi kamus untuk menyimpan nilai cost terkecil untuk
mencapai suatu node
    cost_so_far = {}
    # Memasukkan titik awal dengan cost 0 ke dalam kamus cost_so_far
    cost_so_far[start] = 0

    # Inisialisasi kamus untuk menyimpan jalur terpendek untuk
mencapai suatu node
    came_from = {}

    # Melakukan pencarian
    while heap:
        # Mengambil node dengan cost terkecil dari heap
        (cost, current) = heapq.heappop(heap)

        # Jika node saat ini merupakan tujuan, maka pencarian
selesai
        if current == goal:
            break

        # Menandai node saat ini sebagai sudah dieksplorasi
        explored.add(current)

        # Mengeksplorasi tetangga dari node saat ini
        for move in graf.getmoves(current):
            next_node = move
            move_cost = graf.getBobot()
            cost_to_next_node = move_cost[current][move]
```

```
            # Menghitung total cost untuk mencapai next_node melalui
current
            new_cost = cost_so_far[current] + cost_to_next_node

            # Jika next_node belum pernah dieksplorasi atau memiliki
cost lebih kecil, update cost_so_far dan heap
            if next_node not in cost_so_far or new_cost <
cost_so_far[next_node]:
                cost_so_far[next_node] = new_cost
                priority = new_cost
                heapq.heappush(heap, (priority, next_node))
                came_from[next_node] = current

    # Jika goal tidak dapat dicapai, kembalikan None
    if goal not in came_from:
        return None

    # Membalikan jalur terpendek dari start ke goal
    path = [goal]
    while goal != start:
        goal = came_from[goal]
        path.append(goal)
    path.reverse()

    return path
```

6. AlunAlunBandung.txt

```
jumalahNode = 9
latitude = -6.921676034108906
longitude = 107.60696045260143

listKoordinat = [(-6.922494916145625, 107.60763931601781),
                 (-6.921223970382415, 107.60778905040722),
                 (-6.921053385937518, 107.60646713138735),
                 (-6.920900181593247, 107.6050873393599),
                 (-6.923430322256907, 107.60630669044276),
                 (-6.9224317071220955, 107.60640087724039),
                 (-6.923096728210059, 107.60392863424624),
                 (-6.922099752101955, 107.60401037846468),
                 (-6.920824269765021, 107.60408770507536)
                 ]

listNode = ["Jalan Dalem Kaun - Alun-Alun Timur","Jalan Asia Afrika
- Alun-Alun Timur","Jalan Asia Afrika - Banceuy","Jalan Asia Afrika
- Alkateri","Jalan Kepatihan - Dewi Sartika","Jalan Dewi Sartika -
Dalem Kaun","Jalan Kepatihan - Otto Iskandar Dinata - Karang
Anyar","Jalan Cibadak - Otto Iskandar Dinata - Dalem Kaun","Jalan
Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika"]

matrix = [(0,1,0,0,0,1,0,0,0),
              (1,0,1,0,0,0,0,0,0),
              (0,1,0,1,0,0,0,0,0),
              (0,0,1,0,0,0,0,0,1),
              (0,0,0,0,0,1,1,0,0),
              (1,0,0,0,1,0,0,1,0),
```

```
                (0,0,0,0,1,0,0,1,0),
                (0,0,0,0,0,1,1,0,1),
                (0,0,0,1,0,0,0,1,0)
]
```

7. MakamBungKarno.txt

```
jumalahNode = 9
latitude = -8.084819
longitude = 112.176346
listKoordinat = [(-8.086586, 112.174459),
                 (-8.085713, 112.172071),
                 (-8.087059, 112.178494),
                 (-8.083405, 112.176965),
                 (-8.082984, 112.175750),
                 (-8.088774, 112.178215),
                 (-8.088249, 112.174025),
                 (-8.087432, 112.171255),
                 (-8.085798, 112.174862)]

listNode = ["Jalan Ir. Soekarno-Sumantri Brojonegoro",
            "Jalan Sumantri Brojonegoro-Cakraningrat",
            "Jalan Kalasan-Borobudur",
            "Jalan Ir. Soekarno-Borobudur",
            "Jalan Sentot Prawirodirjo-Antasari",
            "Jalan Dieng-Borobudur",
            "Jalan Dewi Sartika-Ir. Soekarno",
            "Jalan Cakraningrat-Dewi Sartika",
            "Jalan Ir. Soekarno-Kalasan"]


matrix = [(0,1,0,0,0,0,1,0,1),
          (1,0,0,0,1,0,0,1,0),
          (0,0,0,1,0,1,0,0,1),
          (0,0,1,0,1,0,0,0,1),
          (0,1,0,1,0,0,0,0,1),
          (0,0,1,0,0,0,0,0,0),
          (1,0,0,0,0,0,0,1,0),
          (0,1,0,0,0,0,1,0,0),
          (1,0,1,1,1,0,0,0,0)]
```

8. MapItb.txt

```
jumalahNode = 16
latitude = -6.89293
longitude = 107.611009
listKoordinat = [(-6.892615, 107.610427),
                 (-6.891919, 107.610392),
                 (-6.891360, 107.611044),
                 (-6.891022, 107.611039),
                 (-6.891035, 107.609718),
                 (-6.892633, 107.608776),
                 (-6.891045, 107.608679),
                 (-6.891046, 107.608193),
```

```
                (-6.890972, 107.611548),
                (-6.891327, 107.612175),
                (-6.890967, 107.612101),
                (-6.889843, 107.611538),
                (-6.889918, 107.609011),
                (-6.888740, 107.609041),
                (-6.888710, 107.611514),
                (-6.890107, 107.608222)]

listNode = ["Jalan A-I","Jalan A-B","Jalan B-VI","Jalan B-II
Timur","Jalan B-II Barat","Jalan C-D-I","Jalan C-II","Jalan
D-II","Jalan G-II","Jalan I-VI","Jalan H-I-II","Jalan G-IV","Jalan
E-III-IV","Jalan E-V","Jalan G-V","Jalan D-III"]

matrix = [(0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0),
                  (1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0),
                  (0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0),
                  (0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0),
                  (0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0),
                  (1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0),
                  (0,0,0,0,1,1,0,1,0,0,0,0,0,0,0,0),
                  (0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1),
                  (0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0),
                  (1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0),
                  (0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0),
                  (0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0),
                  (0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1),
                  (0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0),
                  (0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0),
                  (0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0)
                 ]
```

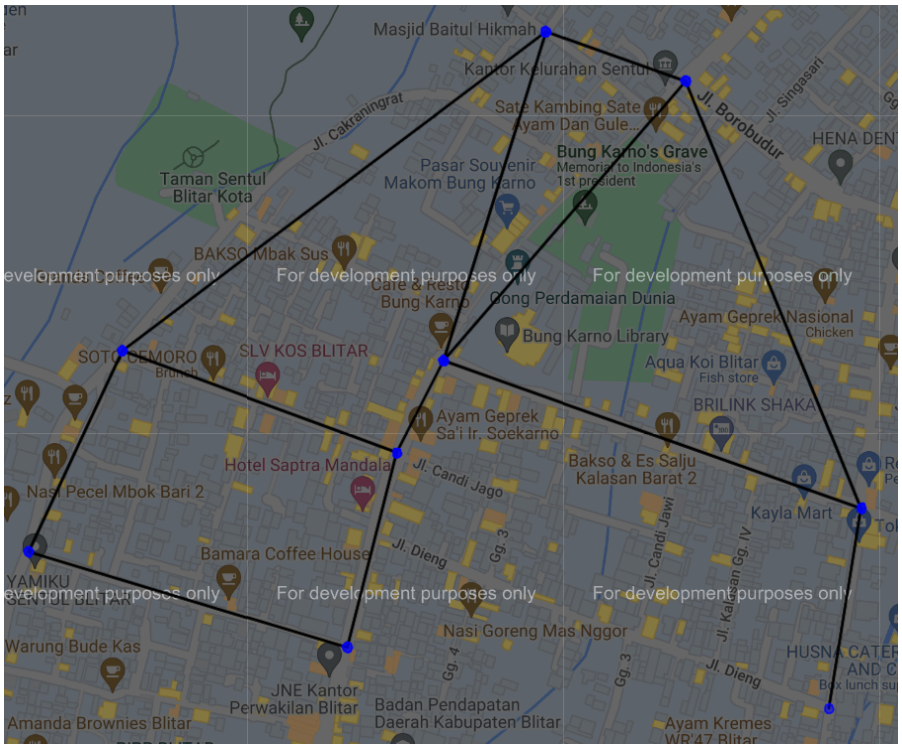
BAB 3
INPUT DAN OUTPUT PROGRAM

1. Peta/graf input

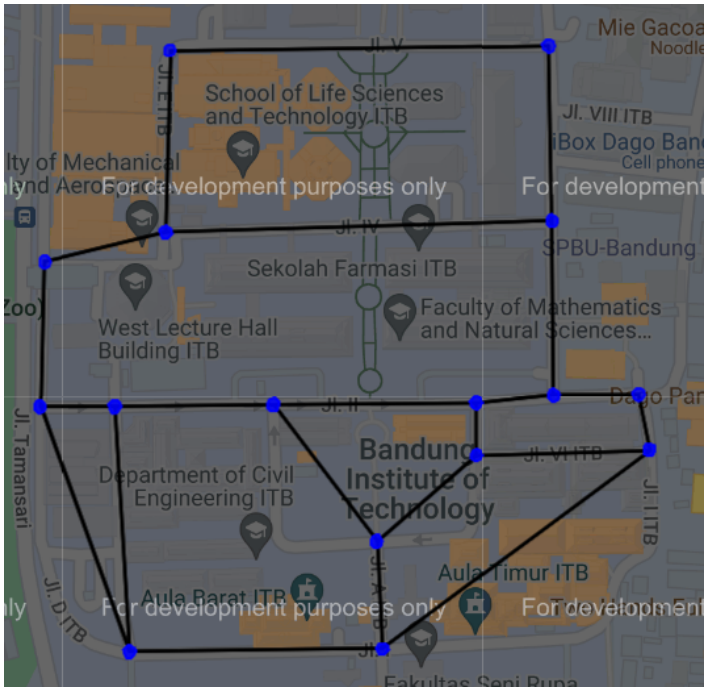
Berikut adalah tampilan graf dan peta input yang akan digunakan:

- AlunAlunBandung.txt

- MakamBungKarno.txt



- MapItb.txt



2. Output screenshot peta

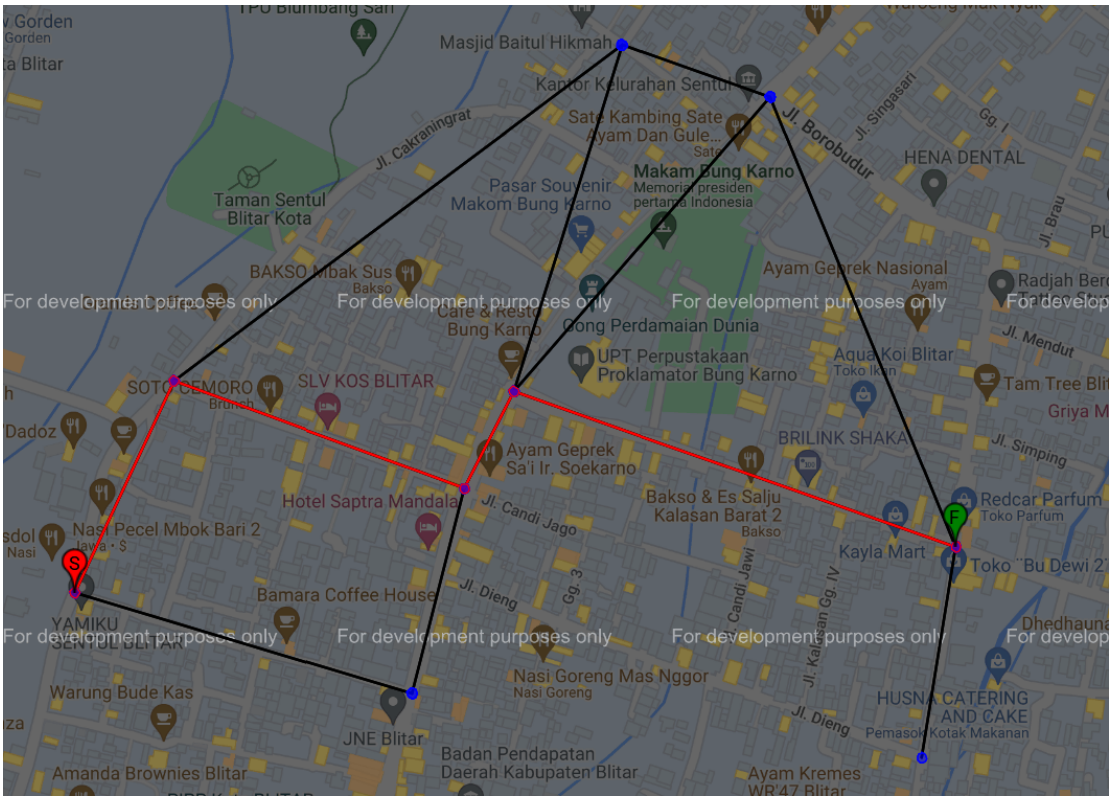
Berikut adalah beberapa hasil lintasan terpendek antara dua buah simpul yang diperoleh dari graf masukan:

a. Output 1

MakamBungKarno.txt

```
jumalahNode = 9
latitude = -8.084819
longitude = 112.176346
listKoordinat = [(-8.086586, 112.174459),
                 (-8.085713, 112.172071),
                 (-8.087059, 112.178494),
                 (-8.083405, 112.176965),
                 (-8.082984, 112.175750),
                 (-8.088774, 112.178215),
                 (-8.088249, 112.174025),
                 (-8.087432, 112.171255),
                 (-8.085798, 112.174862)]

listNode = ["Jalan Ir. Soekarno-Sumantri Brojonegoro",
            "Jalan Sumantri Brojonegoro-Cakraningrat",
            "Jalan Kalasan-Borobudur",
            "Jalan Ir. Soekarno-Borobudur",
            "Jalan Sentot Prawirodirjo-Antasari",
            "Jalan Dieng-Borobudur",
            "Jalan Dewi Sartika-Ir. Soekarno",
            "Jalan Cakraningrat-Dewi Sartika",
            "Jalan Ir. Soekarno-Kalasan"]


matri… = [(0,1,0,0,0,0,1,0,1),
          (1,0,0,0,1,0,0,1,0),
          (0,0,0,1,0,1,0,0,1),
          (0,0,1,0,1,0,0,0,1),
          (0,1,0,1,0,0,0,0,1),
          (0,0,1,0,0,0,0,0,0),
          (1,0,0,0,0,0,0,1,0),
          (0,1,0,0,0,0,1,0,0),
          (1,0,1,1,1,0,0,0,0)]
```

```
(-8.088774, 112.178215)
(-8.088249, 112.174025)
(-8.087432, 112.171255)
(-8.085798, 112.174862)

Adjacency Matrix:
(0, 1, 0, 0, 0, 0, 1, 0, 1)
(1, 0, 0, 0, 1, 0, 0, 1, 0)
(0, 0, 0, 1, 0, 1, 0, 0, 1)
(0, 0, 1, 0, 1, 0, 0, 0, 1)
(0, 1, 0, 1, 0, 0, 0, 0, 1)
(0, 0, 1, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 0, 0, 0, 0, 1, 0)
(0, 1, 0, 0, 0, 0, 1, 0, 0)
(1, 0, 1, 1, 1, 0, 0, 0, 0)

Nama Jalan :
> Jalan Ir. Soekarno-Sumantri Brojonegoro
> Jalan Sumantri Brojonegoro-Cakraningrat
> Jalan Kalasan-Borobudur
> Jalan Ir. Soekarno-Borobudur
> Jalan Sentot Prawirodirjo-Antasari
> Jalan Dieng-Borobudur
> Jalan Dewi Sartika-Ir. Soekarno
> Jalan Cakraningrat-Dewi Sartika
> Jalan Ir. Soekarno-Kalasan

Masukkan Start (Nama Jalan) : Jalan Kalasan-Borobudur
Masukkan Tujuan (nama jalan) : Jalan Cakraningrat-Dewi Sartika
1. A*
2. UCS
Pilih algoritma yang digunakan (1 atau 2) : 1

Rute Terpendeknya adalah :
Jalan Kalasan-Borobudur -> Jalan Ir. Soekarno-Kalasan -> Jalan Ir. Soekarno-Sumantri Brojoneg
oro -> Jalan Sumantri Brojonegoro-Cakraningrat -> Jalan Cakraningrat-Dewi Sartika
Jarak terpendek dari Jalan Kalasan-Borobudur menuju Jalan Cakraningrat-Dewi Sartika adalah 1013
.3751101598347 meter.


Peta telah disimpan dalam file output-astar.html di folder yang sama dengan program ini.
```



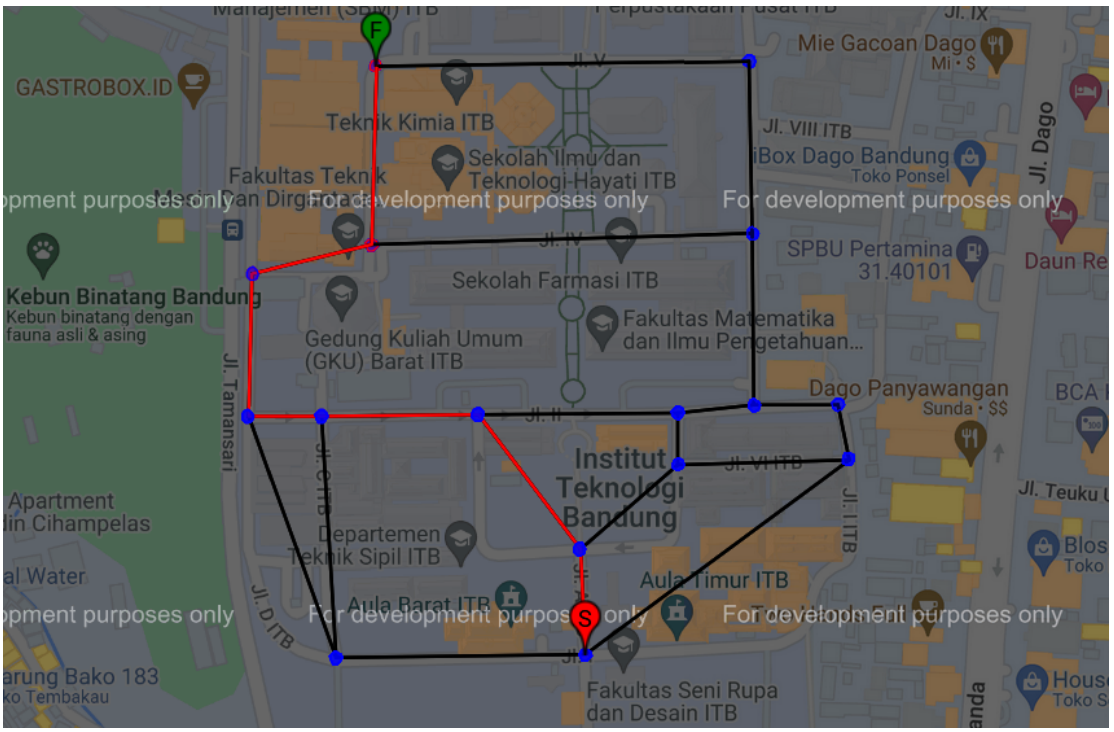
Output ini menampilkan lintasan terpendek dari Jalan Kalasan-Borobudur ke Jalan Cakraningrat-Dewi Sartika

b. Output 2

MapItb.txt

```
jumlahNode = 16
latitude = -6.89293
longitude = 107.611009
listKoordinat = [(-6.892615, 107.610427),
                 (-6.891919, 107.610392),
                 (-6.891360, 107.611044),
                 (-6.891022, 107.611039),
                 (-6.891035, 107.609718),
                 (-6.892633, 107.608776),
                 (-6.891045, 107.608679),
                 (-6.891046, 107.608193),
                 (-6.890972, 107.611548),
                 (-6.891327, 107.612175),
                 (-6.890967, 107.612101),
                 (-6.889843, 107.611538),
                 (-6.889918, 107.609011),
                 (-6.888740, 107.609041),
                 (-6.888710, 107.611514),
                 (-6.890107, 107.608222)]

listNode = ["Jalan A-I",
"Jalan A-B",
"Jalan B-VI",
"Jalan B-II Timur",
"Jalan B-II Barat",
"Jalan C-D-I",
"Jalan C-II",
"Jalan D-II",
"Jalan G-II",
"Jalan I-VI",
"Jalan H-I-II",
"Jalan G-IV",
"Jalan E-III-IV",
"Jalan E-V",
"Jalan G-V",
"Jalan D-III"]

matrix = [(0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0),
                 (1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0),
                 (0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0),
                 (0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0),
                 (0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0),
                 (1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0),
                 (0,0,0,0,1,1,0,1,0,0,0,0,0,0,0,0),
                 (0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1),
                 (0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0),
                 (1,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0),
                 (0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0),
                 (0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0),
                 (0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1),
                 (0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0),
                 (0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0),
                 (0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0)
                 ]
```

```
(0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1)
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0)
(1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0)
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)

Nama Jalan :
> Jalan A-I
> Jalan A-B
> Jalan B-VI
> Jalan B-II Timur
> Jalan B-II Barat
> Jalan C-D-I
> Jalan C-II
> Jalan D-II
> Jalan G-II
> Jalan I-VI
> Jalan H-I-II
> Jalan G-IV
> Jalan E-III-IV
> Jalan E-V
> Jalan G-V
> Jalan D-III

Masukkan Start (Nama Jalan) : Jalan A-I
Masukkan Tujuan (nama jalan) : Jalan E-V
1. A*
2. UCS
Pilih algoritma yang digunakan (1 atau 2) : 2

Rute Terpendeknya adalah :
Jalan E-V -> Jalan E-III-IV -> Jalan D-III -> Jalan D-II -> Jalan C-II -> Jalan B-II Barat
 -> Jalan A-B -> Jalan A-I
Jarak terpendek dari Jalan A-I menuju Jalan E-V adalah 694.210810002344 meter.


Peta telah disimpan dalam file output-ucs.html di folder yang sama dengan program ini.
```



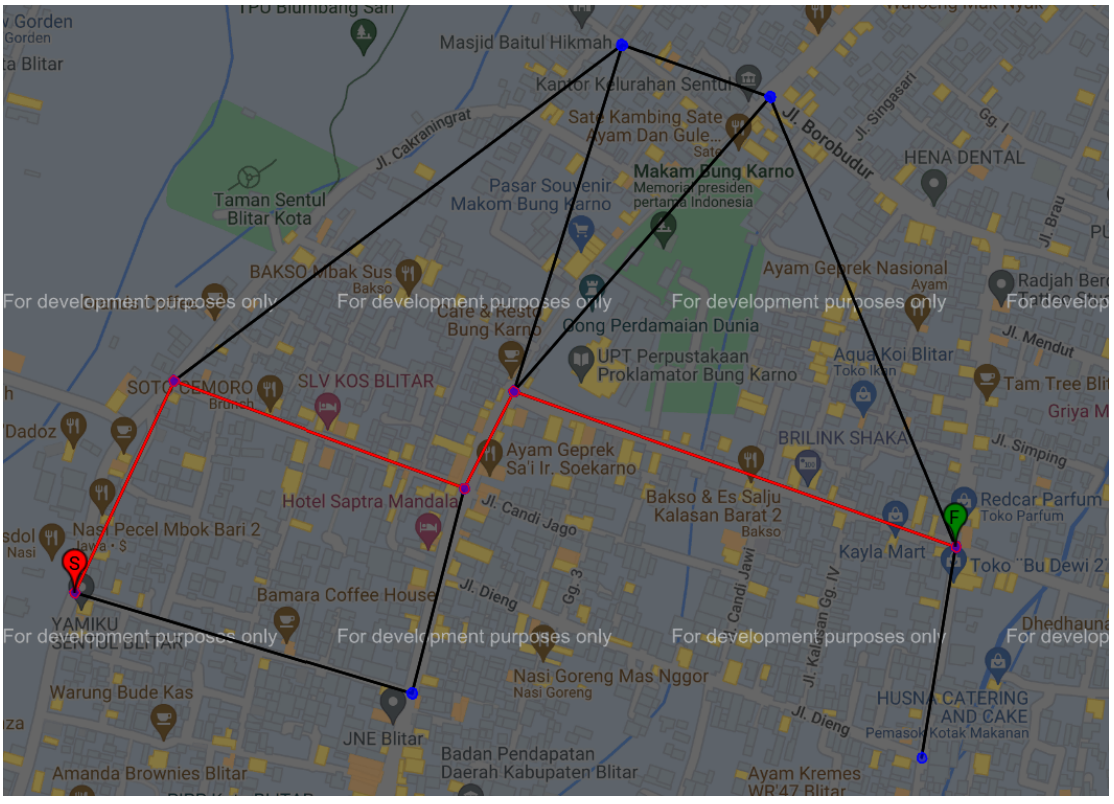
Output ini menampilkan lintasan terpendek dari Jalan A-I ke Jalan E-V

c. Output 3

AlunAlunBandung.txt

```
jumalahNode = 9
latitude = -6.921676034108906
longitude = 107.60696045260143

listKoordinat = [(-6.922494916145625, 107.60763931601781),
                 (-6.921223970382415, 107.60778905040722),
                 (-6.921053385937518, 107.60646713138735),
                 (-6.920900181593247, 107.6050873393599),
                 (-6.923430322256907, 107.60630669044276),
                 (-6.9224317071220955, 107.60640087724039),
                 (-6.923096728210059, 107.60392863424624),
                 (-6.922099752101955, 107.60401037846468),
                 (-6.920824269765021, 107.60408770507536)
                 ]

listNode = ["Jalan Dalem Kaun - Alun-Alun Timur",
"Jalan Asia Afrika - Alun-Alun Timur",
"Jalan Asia Afrika - Banceuy",
"Jalan Asia Afrika - Alkateri",
"Jalan Kepatihan - Dewi Sartika",
"Jalan Dewi Sartika - Dalem Kaun",
"Jalan Kepatihan - Otto Iskandar Dinata - Karang Anyar",
"Jalan Cibadak - Otto Iskandar Dinata - Dalem Kaun",
"Jalan Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika"]

matrix = [(0,1,0,0,0,1,0,0,0),
          (1,0,1,0,0,0,0,0,0),
          (0,1,0,1,0,0,0,0,0),
          (0,0,1,0,0,0,0,0,1),
          (0,0,0,0,0,1,1,0,0),
          (1,0,0,0,1,0,0,1,0),
          (0,0,0,0,1,0,0,1,0),
          (0,0,0,0,0,1,1,0,1),
          (0,0,0,1,0,0,0,1,0)
]
```

```
(-6.921053385937518, 107.60646713138735)
(-6.920900181593247, 107.6050873393599)
(-6.923430322256907, 107.60630669044276)
(-6.9224317071220955, 107.60640087724039)
(-6.923096728210059, 107.60392863424624)
(-6.922099752101955, 107.60401037846468)
(-6.920824269765021, 107.60408770507536)

Adjacency Matrix:
(0, 1, 0, 0, 0, 1, 0, 0, 0)
(1, 0, 1, 0, 0, 0, 0, 0, 0)
(0, 1, 0, 1, 0, 0, 0, 0, 0)
(0, 0, 1, 0, 0, 0, 0, 0, 1)
(0, 0, 0, 0, 0, 1, 1, 0, 0)
(1, 0, 0, 0, 1, 0, 0, 1, 0)
(0, 0, 0, 0, 1, 0, 0, 1, 0)
(0, 0, 0, 0, 0, 1, 1, 0, 1)
(0, 0, 0, 1, 0, 0, 0, 1, 0)

Nama Jalan :
> Jalan Dalem Kaun - Alun-Alun Timur
> Jalan Asia Afrika - Alun-Alun Timur
> Jalan Asia Afrika - Banceuy
> Jalan Asia Afrika - Alkateri
> Jalan Kepatihan - Dewi Sartika
> Jalan Dewi Sartika - Dalem Kaun
> Jalan Kepatihan - Otto Iskandar Dinata - Karang Anyar
> Jalan Cibadak - Otto Iskandar Dinata - Dalem Kaun
> Jalan Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika

Masukkan Start (Nama Jalan) : Jalan Dalem Kaun - Alun-Alun Timur
Masukkan Tujuan (nama jalan) : Jalan Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika
1. A*
2. UCS
Pilih algoritma yang digunakan (1 atau 2) : 2

Rute Terpendeknya adalah :
Jalan Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika ->  Jalan Cibadak - Otto Iskandar Din
ata - Dalem Kaun ->  Jalan Dewi Sartika - Dalem Kaun ->  Jalan Dalem Kaun - Alun-Alun Timur
Jarak terpendek dari Jalan Dalem Kaun - Alun-Alun Timur menuju Jalan Jend. Sudirman - Otto Iska
ndar Dinata - Asia Afrika adalah 545.4114866905334 meter.


Peta telah disimpan dalam file output-ucs.html di folder yang sama dengan program ini.
```



Output ini menampilkan lintasan terpendek dari Jalan Dalem Kaun - Alun-Alun Timur ke Jalan Jend. Sudirman - Otto Iskandar Dinata - Asia Afrika

BAB 4
KESIMPULAN DAN KOMENTAR

1. Kesimpulan

Algoritma A* dan UCS mempunyai kelebihan dan kekurangan masing-masing dalam menyelesaikan masalah mencari jalur terpendek pada graf. A* mengombinasikan pendekatan heuristik dengan algoritma pencarian graf untuk menghasilkan jalur terpendek dengan efektif dan efisien. A* juga dapat menangani masalah jalur yang kompleks dan memiliki banyak cabang dengan mengevaluasi setiap jalur dan memilih jalur dengan biaya terendah. Meskipun demikian, pada kasus graf yang sangat besar dan kompleks, A* mungkin tidak selalu menghasilkan solusi optimal.

Di sisi lain, algoritma UCS menggunakan pendekatan biaya tanpa mempertimbangkan informasi heuristik, sehingga lebih cocok untuk masalah yang memiliki graf yang lebih kecil dan memiliki biaya yang jelas. Kelebihan dari algoritma UCS adalah kemampuannya untuk menemukan jalur terpendek secara optimal dan bisa digunakan pada masalah graf dengan biaya yang berbeda. Namun, kekurangan dari UCS adalah kinerjanya kurang optimal pada graf yang kompleks dan memiliki banyak cabang.

2. Komentar

“Tugasnya mantap” - raihan

“Punggung sakit” - ilham

LAMPIRAN

1. Pranala Github

Berikut adalah tautan repositori dari program yang telah dibuat:

https://github.com/raihaniqbal24/Tucil3_13518134_13521068

2. Checklist Kelengkapan

No	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	✔	