TUGAS KECIL III IF2211

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Disusun Oleh :

Almeiza Arvin Muzaki	13519066
Muhammad Naufal Izza Fikry	13519088

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

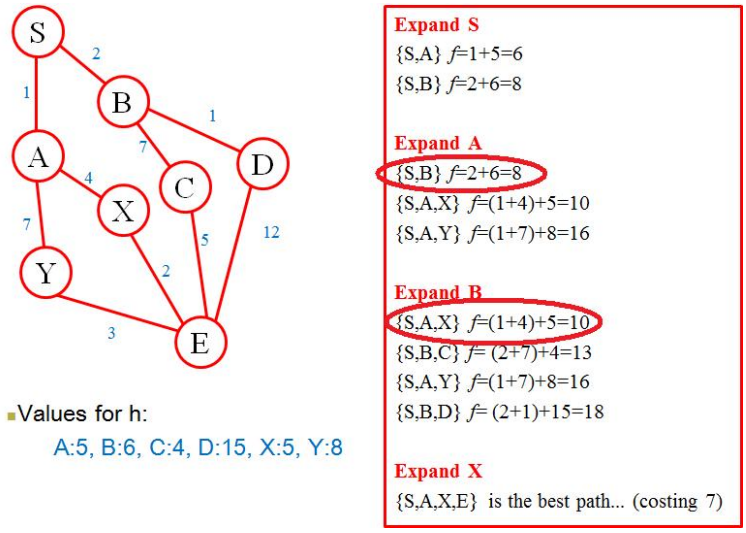
2021

Bab 1
Deskripsi Masalah dan Dasar Teori

Dalam persoalan pencarian jalur paling optimal, sampai saat ini, telah banyak dilakukan pengkajian algoritma terbaik. Ada banyak algoritma pencarian untuk graf yang telah dirumuskan dan masing-masing memiliki keunggulan dan kelemahannya tersendiri.

Setiap algoritma pencarian memiliki fungsi heuristik evaluasi yang dituliskan sebagai f(n).

Salah satu algoritma pencarian yang cukup populer dalam dunia pemrograman adalah A*. Jika sebelumnya telah dikenal Algoritma Uniform Best Search dan Greedy Best Search, maka A* star adalah algoritma yang menggabungkan fungsi heuristik keduanya. Dengan g(n) merupakan current cost untuk mencapai suatu titik n yang diperoleh dari Uniform Cost Search, dan h(n) merupakan perkiraan total cost dari titik n terhadap goal yang diperoleh dari Greedy Best Search, maka f(n) dari A* = g(n) + h(n).



Gambar 1.2. Contoh Kasus Sederhana dengan Algoritma A*

Pada contoh di atas, pencarian dimulai dari simpul S. Ekspansi dilakukan pada A dan B. Karena f(A) < f(B), ekspansi dilakukan ke A dan f(A) dihapus dari queue. Karena f(B) saat ini nilainya paling kecil, ekspansi B dan hapus B dari queue. Proses tersebut dilakukan terus-menerus berulang-ulang hingga ditemukan node tujuan yaitu node E.

Bab II
Penyelesaian dan Hasil

Untuk menyelesaikan persoalan ini, kami menggunakan bahasa Python dengan bantuan Jupyter Notebook sebagai environmentnya. Seluruh program terletak pada directory src yang terdiri dari program utama (Tucil3.ipynb) dan file-file pendukung lainnya (algorithm.py, graph.py, helper.py, lab.py, node.py dan prio_queue.py).

Input program menggunakan file eksternal .txt yang terdiri dari 4 jenis daerah:

1. Daerah sekitar kampus ITB Ganesa (KoorITB, AdjITB)
2. Daerah sekitar Alun-Alun Bandung (KoorAlunalun, AdjAlunalun)
3. Daerah sekitar Buah Batu, Bandung (KoorBubat, AdjBubat)
4. Daerah sekitar Mojosari, Mojokerto (KoorMojosari, AdjMojosari)

File Koor(daerah) berisi daftar koordinat persimpangan pada terkait, sedangkan File Adj(daerah) berisi adjacency matriks setiap persimpangan.
Inisialisasi map bergantung pada File Koor dan File Adj yang diinput. File View(daerah) berisi koordinat daerah, digunakan untuk inisialisasi Map sesuai nama daerah yang tercantum pada File Koor dan FileAdj yang sebelumnya telah diinputkan.

Tucil3.ipynb (main program)

```
import math as m
import folium as fl
from helper import distance
from node import Node
from graph import Graph
from algorithm import a_star

# def read_data():
#     Graph.read_data()

def init_map():
    Map = fl.Map(location=Graph.starting_view_coordinate, width=800, height=512, zoom_start=13)
    return Map

#////////////////////////

Node.reset()
Graph.read_data()
Map = init_map()
```

```
# Drawing Base
for i in range(0, len(Graph.nameList)):
    fl.Marker(
        location=[Graph.latList[i], Graph.lngList[i]],
        popup=str(Node.list_node[i].id)+" - "+Graph.nameList[i],
        icon=fl.Icon(color="blue"),
    ).add_to(Map)


point1=[]
point2=[]
for i in range (0, len(Graph.adjmatrix)):
    for j in range (0, len(Graph.adjmatrix[i])):
        if(Graph.adjmatrix[i][j]!='0'):
            point1= [Graph.latList[i], Graph.lngList[i]]
            point2= [Graph.latList[j], Graph.lngList[j]]
            Graph.edgesList+=[[point1, point2]]
        point1=[]
        point2=[]


for i in Graph.edgesList:
    fl.PolyLine(i, color='grey').add_to(Map)

Map
```

```
for i in Graph.edgesList:
    fl.PolyLine(i, color='grey').add_to(Map)

if len(final_path) > 0:
    #- adding final_path to map
    EdgesList = []
    for i in range(len(final_path)-1):
        point1 = [final_path[i].lat, final_path[i].lng]
        point2 = [final_path[i+1].lat, final_path[i+1].lng]
        EdgesList+=[[point1, point2]]
        point1=[]
        point2=[]

    for i in EdgesList:
        fl.PolyLine(i, color='blue').add_to(Map)

Map
```

graph.py (Memuat class graph beserta seluruh atribut dan method terkait graph)

```
from helper import distance
from node import Node
FILE_PATH = "../test/"

class Graph:
    nodesList=[]
    edgesList=[]
    adjmatrix=[]
    latList=[]
    lngList=[]
    nameList=[]
    location=""
    filename = ""
    starting_view_coordinate = []

    def __init__(self):
        pass

    def AddNode(self, Lat, Lng):
        if[Lat, Lng] not in Graph.nodesList:
            Graph.nodesList+=[[Lat, Lng]]
            Graph.latList+=[Lat]
            Graph.lngList+=[Lng]

    def read_data():
        # MENU
        supported_area = []
        input_valid = False
        selected_area = ""

        with open(FILE_PATH+"supported_area.txt", "r") as f:
            supported_area = f.read().split('\n')
        while not input_valid:
            print("Pilih Daerah:")
            for i in range(len(supported_area)):
                print("[%d] %s"%(i+1, supported_area[i]))
            x = int(input(" >> "))
            if 0 < x and x <= len(supported_area):
                input_valid = True
            else:
                print("Masukan tidak valid.")
        selected_area = str(supported_area[x-1])


        # Baca Koor
        Graph.location = "Koor"+selected_area
        Graph.filename = FILE_PATH+selected_area+'/'+Graph.location+".txt"
        f = open(Graph.filename, "r")
        file_contents = f.read().splitlines()
```

```
        f.close()

        temp=[]
        for i in file_contents:
            temp+=i.split()

        for i in range (0, len(temp)):
            if i%3==0:
                Graph.latList+=[float(temp[i])]
            elif i%3==1:
                Graph.lngList+=[float(temp[i])]
            elif i%3==2:
                Graph.nameList+=[temp[i]]

        path = Graph.filename
        with open(path, 'r') as f:
            nodes = f.read().split('\n')
            for i in range(len(nodes)):
                nodes[i] = nodes[i].split(' ')
            for node in nodes:
                Node(str(node[2]), float(node[0]), float(node[1]))


        # Baca Adj
        AdjFile = "Adj"+selected_area
        filename2=FILE_PATH+selected_area+'/'+AdjFile+".txt"
        f = open(filename2, "r")
        file_contents2 = f.read().splitlines()
        f.close()

        for i in file_contents2:
            i=i.split()
        for i in range (0, len(file_contents2)):
            Graph.adjmatrix+=[file_contents2[i].split()]


        for i in range (0, len(Graph.adjmatrix)):
            for j in range (0, len(Graph.adjmatrix[i])):
                if(Graph.adjmatrix[i][j]=='1'):
                    Graph.adjmatrix[i][j]=distance(Graph.latList[i], Graph.lngList[i], Graph.latList[j],
Graph.lngList[j])


        # Baca starting_view_coordinate
        ViewFile = "View"+selected_area
        with open(FILE_PATH+selected_area+'/'+ViewFile+".txt", "r") as f:
            start_koor = f.read().split(' ')
            for i in range(len(start_koor)):
             start_koor[i] = float(start_koor[i])
            Graph.starting_view_coordinate = start_koor
```

node.py (berisi class Node dan atribut-method pendukungnya)

```
from helper import distance
class Node():
    INFINITY_R = -1
    list_node = []
    node_count = 0

    start_node_id = None
    end_node_id = None

    list_distance = [] # list jarak tiap node ke end_node

    def __init__(self, name, lat, lng):
        Node.node_count += 1
        self.id = int(Node.node_count)
        self.name = name
        self.lat = lat
        self.lng = lng

        self.prev_distance = Node.INFINITY_R
        self.prev_node = None

        Node.list_node.append(self)

    def __del__(self):
        # print("node %d destroyed"%(self.id))
        pass

    def set_start(id):
        valid = False
        for node in Node.list_node:
            if node.id == id:
                valid = True
                break
        if valid:
            Node.start_node_id = int (id)
        else:
            print("Invalid Node!")

    def set_end(id):
        # jika valid, melakukan set end_node dan menghitung jarak tiap-tiap node ke end_node
        valid = False
        for node in Node.list_node:
```

```
            if node.id == id:
                valid = True
                break
        if valid:
            Node.end_node_id = int (id)
            Node.calculate_distances()
        else:
            print("Invalid Node!")

    def update_prev_distance(self, prev_distance):
        self.prev_distance = float(prev_distance)

    def update_prev_node(self, prev_node_reference):
        self.prev_node = prev_node_reference

    def get_start_node():
        if (Node.start_node_id == None):
            return None
        ret = None
        for node in Node.list_node:
            if node.id == Node.start_node_id:
                ret = node
                break
        return ret

    def get_end_node():
        if (Node.end_node_id == None):
            return None
        ret = None
        for node in Node.list_node:
            if node.id == Node.end_node_id:
                ret = node
                break
        return ret

    def get_index(self):
        return int(self.id-1)

    def get_node(id):
        i = 0
        while i<len(Node.list_node):
            if(Node.list_node[i].id == id):
                return Node.list_node[i]
            i+=1
        return None

    def set_start_end_handler():
        print()
        print("              --=Set start-end=--              ")
        print("Klik 'i' pada peta untuk mengetahui nama persimpangan")
```

```
    Node.print_all()
    Node.set_start(int(input("Start: ")))
    Node.set_end(int(input("End: ")))


def calculate_distances():
    end_node = Node.get_end_node()
    for node in Node.list_node:
        d = distance(node.lat, node.lng, end_node.lat, end_node.lng)
        Node.list_distance.append(d)

def print_all():
    i = 0
    for node in Node.list_node:
        i += 1
        print("[%d] %s"%(i,node.name))

def reset():
    i=0
    while i < len(Node.list_node):
        del Node.list_node[i]
    Node.list_node = []
    Node.node_count = 0
    Node.start_node_id = None
    Node.end_node_id = None
    print("reset success")
```

helper.py (berisi fungsi-fungsi pendukung, seperti `distance` yang menggunakan persamaan haversine untuk menghitung jarak 2 titik)

```
import math as m

def distance (Lat1, Lng1, Lat2, Lng2):
    r = 6371 #radius bumi
    Lat1=m.radians((Lat1))
    Lat2=m.radians((Lat2))
    Lng1=m.radians((Lng1))
    Lng2=m.radians((Lng2))
    dlat = (Lat1-Lat2)
    dlng = (Lng1-Lng2)
    return 2*r*(m.asin((m.sqrt(m.sin(dlng/2)**2 + m.cos(Lat1)*m.cos(Lat2)*m.sin(dlat/2)**2))))
```

prio_queue.py (berisi class PriorityQueue dan komponen-komponen pendukungnya)

```
class PriorityQueue():
    def __init__(self):
        self.pq = []

    def put(self, item_tuple):
        try:
            push_last = True
            for i in range(len(self.pq)):
                if (self.pq[i][0] > item_tuple[0]):
                    self.pq.insert(i, item_tuple)
                    push_last = False
                    break
            if push_last:
                self.pq.append(item_tuple)
        except:
            print("input format not satisfied!")

    def get(self):
        return self.pq.pop(0)

    def del_by_key(self, key):
        for i in range(len(self.pq)):
            if self.pq[i][0] == key:
                self.pq.pop(i)
                break

    def check_value(self, value):
        for i in range(len(self.pq)):
            if id(self.pq[i][1]) == id(value):
                return True
        return False
```

algorithm.py (memuat algoritma A*)

```
from prio_queue import PriorityQueue
from node import Node
from graph import Graph
from helper import distance
def a_star():
```

```
    expanded_nodes_id = [] # untuk tracking node mana saja yang sudah diekspan

    Node.set_start_end_handler()

    # _variables containing reference to the real object_
    start_node = Node.get_start_node()
    end_node = Node.get_end_node()
    adj_matrix = Graph.adjmatrix

    # set prev_distance start_node menjadi 0
    start_node.update_prev_distance(0)

    # buat PriorityQueue baru (pq)
    pq = PriorityQueue()

    #[status] start_node seakan-akan masuk pq
    #-- current_node adalah node yang akan diekspan
    current_node = start_node
    #[status] start_node seakan-akan keluar pq (diekspan)

    # helper boolean
    route_exist = True
    all_neighbour_checked = False
    while(current_node.id != end_node.id and not (all_neighbour_checked)):
        # idx untuk mencari node tetangga
        idx = current_node.get_index()
        neighbour = adj_matrix[idx]
        no_neighbour_left = True
        for i in range(len(neighbour)):
            if (neighbour[i]!='0' and (i+1) not in expanded_nodes_id): # jika tetangga dan belum pernah
diekspan
                no_neighbour_left = False
                # TODO: simpan dulu prev_node, prev_distance, dan end_distance ke variabel lokal ds, pn,
de (distance_start, previous_node, distance_end).
                # jika checked_node belum di pq (prev_distance = INFINITY_R), update prev_distance dan
prev_node, lalu tambahkan checked_node ke pq
                # jika sudah di pq dan nilai ds < prev_distance

                #-- di sini, current_node = prev_node dari checked_node
                #-- nantinya, checked_node akan dimasukkan ke pq (PriorityQueue)

                checked_node = Node.get_node(i+1) # simpan neighbour[i] ke checked_node

                # menyimpan ke variabel lokal
                ds = distance(current_node.lat, current_node.lng, checked_node.lat, checked_node.lng) +
```

```
current_node.prev_distance
            pn = current_node
            de = Node.list_distance[checked_node.id-1]

            #-- setelah itu, priority_weight checked_node perlu dihitung
            #-- priority_weight = prev_distance dari checked_node + jarak dari checked_node ke
end_node
            priority_weight = ds + de

            # cek apakah checked_node ada di pq
            exist = pq.check_value(checked_node)

            if not exist:
               # update prev_distance
               checked_node.update_prev_distance(ds)
               # update prev_node
               checked_node.update_prev_node(pn)

               pq.put((priority_weight, checked_node))
            else:
               # jika ds < prev_distance, update checked_node pada pq
               if ds < checked_node.prev_distance:
                  # hapus node lama
                  key = checked_node.prev_distance + de
                  pq.del_by_key(key)

                  # update prev_distance
                  checked_node.update_prev_distance(ds)
                  # update prev_node
                  checked_node.update_prev_node(pn)

                  pq.put((priority_weight, checked_node))


      #-- setelah for loop dilakukan, id current_node akan dicatat di expanded_nodes_id
      #-- setelah itu, elemen terdepan dari pq akan diekspan (diset menjadi current_node)
      expanded_nodes_id.append(current_node.id)
      if no_neighbour_left and len(pq.pq) == 0:
         all_neighbour_checked = True
      else:
         current_node = pq.get()[1]

   if (current_node.id!=end_node.id and (all_neighbour_checked and len(pq.pq)==0)):
      route_exist = False
```

```
    print("Expanded Node:", end=" ")
    print(expanded_nodes_id, end = "")
    if route_exist:
        print(" -> %d [FOUND]"%(end_node.id))
    else:
        print(" -> X [NOT FOUND]")

    if not route_exist:
        print("Tidak ada rute yang menghubungkan kedua titik.")
        return []

    path = []
    while(current_node.prev_node != None):
        path.insert(0, current_node)
        current_node = current_node.prev_node
    path.insert(0, current_node)

    jarak_total = 0
    print()
    for i in range(len(path)):
        prev_distance = 0
        if i!=0:
            prev_distance = path[i-1].prev_distance
        jarak = path[i].prev_distance - prev_distance
        print(path[i].id, '-', path[i].name, "[%f km]"%(jarak))
        jarak_total += jarak

    print()
    print("Jarak total: %f km"%(jarak_total))
    print("Path:", end = " ")
    for i in range(len(path)):
        print("[%d] %s"%(path[i].id,path[i].name), end="")
        if i < len(path)-1:
            print(" -> ", end="")
    return path
```

Pemanfaatan algoritma A* pada program ini terletak pada def a_star() dalam algorithm.py. Secara umum, berikut merupakan pseudocode dari def a_star.py.

```
def a_star():
```

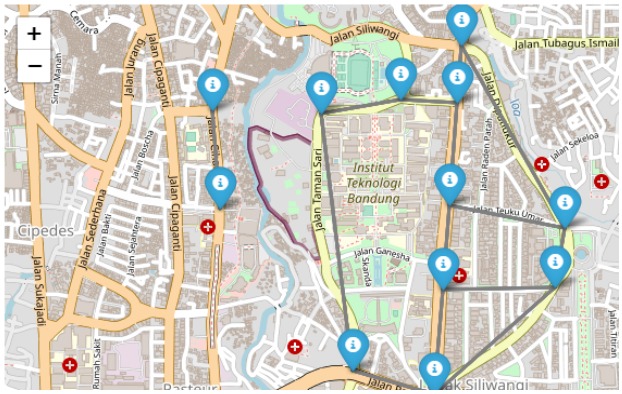
```
    expanded_nodes_id = []
    # untuk tracking node mana saja yang sudah diekspan

    Node.set_start_end_handler()
    #Set node asal dan node tujuan

    # inisiasi variabel
    start_node <- Node.get_start_node()
    end_node <- Node.get_end_node()
    adj_matrix <- Graph.adjmatrix

    #Saat pertama kali, prev_distance start_node adalah 0
    start_node.update_prev_distance(0)

    # buat PriorityQueue baru (pq)
    pq <- PriorityQueue()

    #[status] start_node seakan-akan masuk pq
    #-- current_node adalah node yang akan diekspan
    current_node <- start_node
    #[status] start_node seakan-akan keluar pq (diekspan)

    while(current_node.id != end_node.id) do
        # mencari node tetangga
        idx <- current_node.get_index()
        neighbour <- adj_matrix[idx]

        for i in range(len(neighbour)):
            if (neighbour[i] <> '0' and (i+1) not in expanded_nodes_id):

            # jika tetangga dan belum pernah diekspan, simpan dulu prev_node,
prev_distance, dan end_distance ke variabel lokal ds, pn, de (distance_start,
previous_node, distance_end).
            # jika checked_node belum berada pada pq (prev_distance =
INFINITY_R), update prev_distance dan prev_node, lalu tambahkan checked_node
ke pq
            # jika sudah di pq dan nilai ds < prev_distance

            #-- di sini, current_node = prev_node dari checked_node
            #-- nantinya, checked_node akan dimasukkan ke pq (PriorityQueue)

                checked_node <- Node.get_node(i+1)
                # simpan neighbour[i] ke checked_node

                # menyimpan ke variabel lokal
                ds <- distance(current_node.lat, current_node.lng,
checked_node.lat, checked_node.lng) + current_node.prev_distance
                pn <- current_node
                de <- Node.list_distance[checked_node.id-1]

                #Penghitungan priority_weight checked_node
                #-- priority_weight = prev_distance dari checked_node + jarak
dari checked_node ke end_node (implementasi f(n) = g(n)+h(n)
                priority_weight <- ds + de

                # cek apakah checked_node ada di pq
```

```
                exist <- pq.check_value(checked_node)

                if not exist:
                    # update prev_distance
                    checked_node.update_prev_distance(ds)
                    # update prev_node
                    checked_node.update_prev_node(pn)

                    pq.put((priority_weight, checked_node))
                else:
                    # jika ds < prev_distance, update checked_node pada pq
                    if ds < checked_node.prev_distance:
                        # hapus node lama pada pq berdasarkan keynya
                        key <- checked_node.prev_distance + de
                        pq.del_by_key(key)

                        # update prev_distance
                        checked_node.update_prev_distance(ds)
                        # update prev_node
                        checked_node.update_prev_node(pn)

                        pq.put((priority_weight, checked_node))


        #-- setelah for loop dilakukan, id current_node akan dicatat di
expanded_nodes_id
        #-- setelah itu, elemen terdepan dari pq akan diekspan (diset menjadi
current_node)
        expanded_nodes_id.append(current_node.id)
        current_node <- pq.get()[1]

    #Pengecekan apakah rute berhasil ditemukan
    if (current_node.id <> end_node.id and (all_neighbour_checked and
length(pq.pq)=0)
):
        route_exist <- False

    print("Expanded Node:", end=" ")
    print(expanded_nodes_id, end = "")
    if route_exist:
        print(" -> %d [FOUND]"%(end_node.id))
    else:
        print(" -> X [NOT FOUND]")

    #return [] dan print message apabila rute tidak ditemukan
    if not route_exist:
        print("Tidak ada rute yang menghubungkan kedua titik.")
        return []

    #Pencetakan hasil pencarian dan return path history pencarian   apabila
rute ditemukan
    path = []
    while(current_node.prev_node <> None):
        path.insert(0, current_node)
        current_node <- current_node.prev_node
```

```
    path.insert(0, current_node)

    jarak_total <- 0
    print()
    for i in range(len(path)):
        prev_distance <- 0
        if i <> 0:
            prev_distance <- path[i-1].prev_distance
        jarak <- path[i].prev_distance - prev_distance
        print(path[i].id, '-', path[i].name, "[%f km]"%(jarak))
        jarak_total <- jarak_total + jarak

    print()
    print("Jarak total: %f km"%(jarak_total))
    print("Path:", end = " ")
    for i in range(len(path)):
        print("[%d] %s"%(path[i].id,path[i].name), end="")
        if i < len(path)-1:
            print(" -> ", end="")
    return path
```

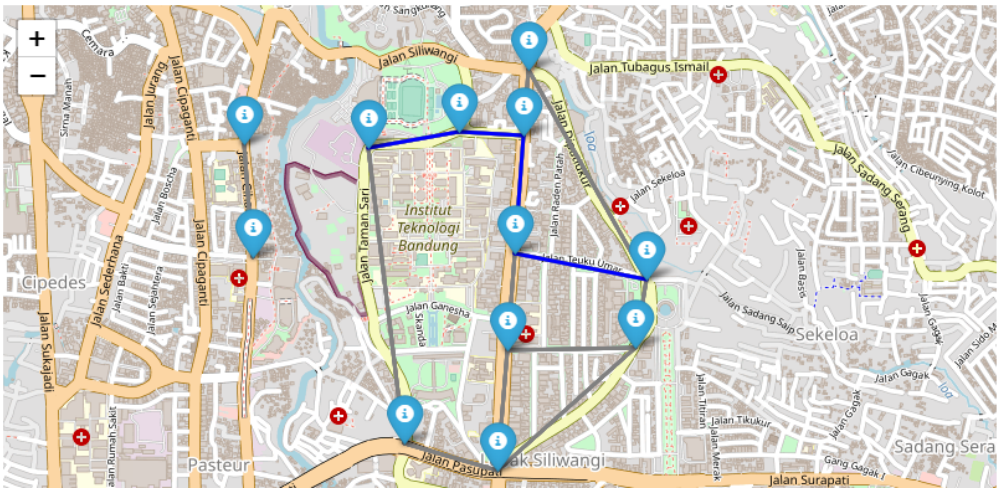
```
                --=Set start-end=--
Klik 'i' pada peta untuk mengetahui nama persimpangan
[1] Dipatiukur-Dago-Siliwangi
[2] DayangSumbi-Dago
[3] Tamansari-DayangSumbi
[4] BelokanTamansariFoodFest
[5] Pasupati-Tamansari
[6] Dago-Pasupati-Dipatiukur
[7] Dago-Dipatiukur-Hasanudin
[8] Dago-Hasanudin
[9] Dipatiukur-Teukuumar
[10] Dago-Teukuumar
[11] Cihampelas-Lamping
[12] Cihampelas-BapaHusen
```

```
Start: 1
End: 11
Expanded Node: [1, 2, 3, 4, 10, 8, 9, 7, 6, 5] -> X [NOT FOUND]
Tidak ada rute yang menghubungkan kedua titik.
```



Gambar 2.1, 2.2. Contoh graf daerah sekitar ITB Ganesa dan kasus pencarian jalur tidak ditemukan

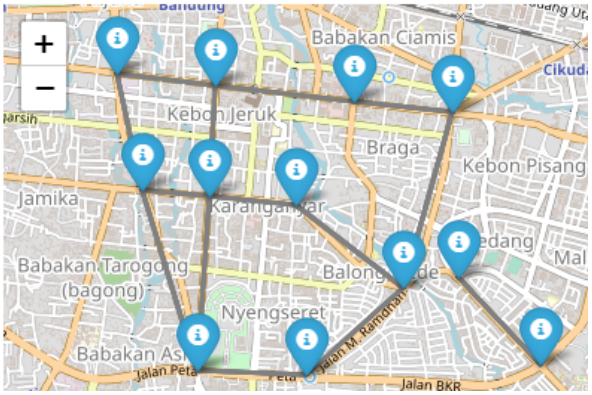
```
Jarak total: 1.569959 km
Path: [4] BelokanTamansariFoodFest -> [3] Tamansari-DayangSumbi -> [2] DayangSumbi-Dago -> [10] Dago-Teukuumar -> [9] Dipatiuku
r-Teukuumar
```



Gambar 2.3. Contoh graf daerah ITB Ganesa dan kasus pencarian jalur ditemukan

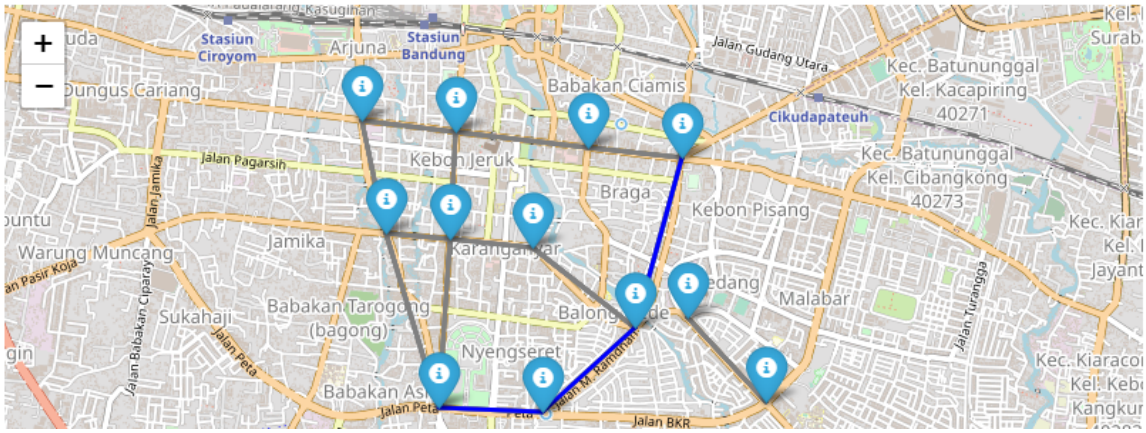
```
--=Set start-end=--
Klik 'i' pada peta untuk mengetahui nama persimpangan
[1] AstanaAnyar-AsiaAfrika
[2] AstanaAnyar-Pasirkoja-Pungkur
[3] Otista-Pungkur
[4] Otista-BKR
[5] BKR-MochRamdan
[6] MochRamdan-Pungkur-Karapitan
[7] Pasundan-Pungkur
[8] AsiaAfrika-Karapitan-Sunda
[9] LengkongBesar-AsiaAfrika
[10] AsiaAfrika-Otista-Sudirman
[11] Bubat-PelajarPejuang
[12] Bubat-Banteng-Gurame
```

```
Start: 4
End: 12
Expanded Node: [4, 5, 6, 3, 7] -> X [NOT FOUND]
Tidak ada rute yang menghubungkan kedua titik.
```



Gambar 2.4, 2.5. Contoh graf daerah sekitar Alun-Alun Bandung dan kasus pencarian jalur tidak ditemukan

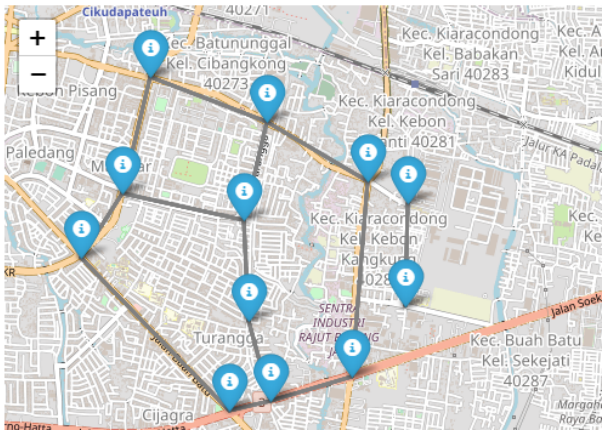
```
4 - Otista-BKR [0.000000 km]
5 - BKR-MochRamdan [0.691854 km]
6 - MochRamdan-Pungkur-Karapitan [0.836524 km]
8 - AsiaAfrika-Karapitan-Sunda [1.160765 km]

Jarak total: 2.689142 km
Path: [4] Otista-BKR -> [5] BKR-MochRamdan -> [6] MochRamdan-Pungkur-Karapitan -> [8] AsiaAfrika-Karapitan-Sunda
```



Gambar 2.6. Contoh graf daerah Alun-Alun Bandung dan kasus pencarian jalur ditemukan

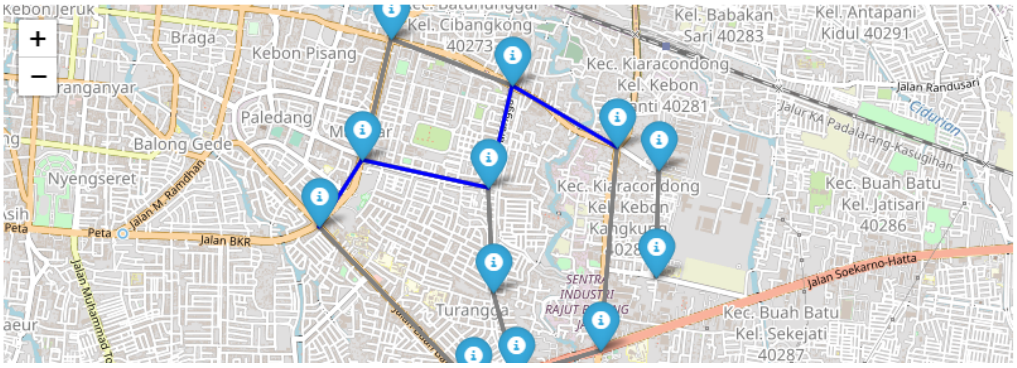
```
                ---=Set start-end=--
Klik 'i' pada peta untuk mengetahui nama persimpangan
[1] Bubat-PelajarPejuang
[2] PelajarPejuang-Martanegara
[3] PelajarPejuang-Gatsu
[4] Bubat-Soetta
[5] Soetta-BatuApi
[6] BatuApi-Salendro
[7] Gatsu-Turangga
[8] Kircon-Gatsu
[9] Soetta-Kircon
[10] Martanegara-Turangga-Salendro
[11] Papanggungan-Gatsu
[12] Papanggungan-Sekejati
```

```
Start: 9
End: 12
Expanded Node: [9, 5, 8, 6, 4, 10, 7, 1, 2, 3] -> X [NOT FOUND]
Tidak ada rute yang menghubungkan kedua titik.
```



Gambar 2.7, 2.8. Contoh graf daerah sekitar Buah Batu, Bandung dan kasus pencarian jalur tidak ditemukan

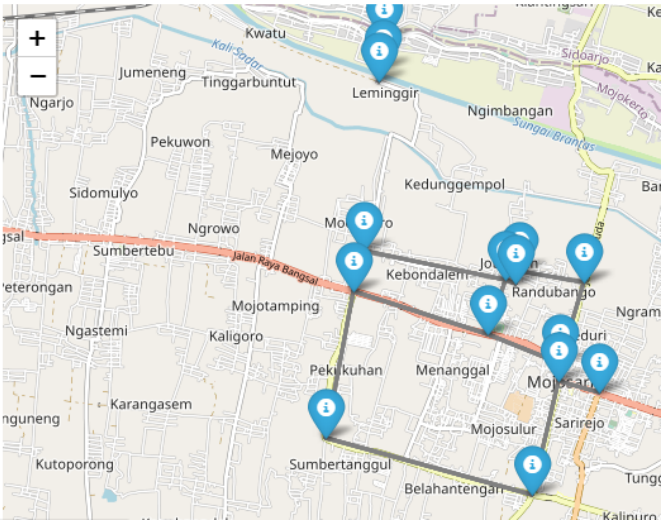
```
Start: 1
End: 8
Expanded Node: [1, 2, 10, 7] -> 8 [FOUND]

1 - Bubat-PelajarPejuang [0.000000 km]
2 - PelajarPejuang-Martanegara [0.592182 km]
10 - Martanegara-Turangga-Salendro [0.980450 km]
7 - Gatsu-Turangga [0.778364 km]
8 - Kircon-Gatsu [0.911532 km]

Jarak total: 3.262527 km
Path: [1] Bubat-PelajarPejuang -> [2] PelajarPejuang-Martanegara -> [10] Martanegara-Turangga-Salendro -> [7] Gatsu-Turangga ->
[8] Kircon-Gatsu
```



Gambar 2.9. Contoh graf daerah Buah Batu, Bandung dan kasus pencarian jalur ditemukan

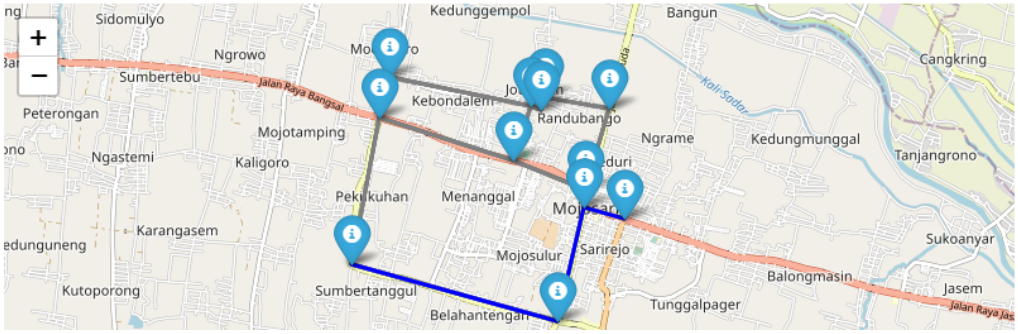
```
                --=Set start-end=--
Klik 'i' pada peta untuk mengetahui nama persimpangan
[1] perempatan_pekukuhan
[2] pertigaan_S.Parman
[3] belokan_A.Yani
[4] pertigaan_GajahMada-RadenWijaya
[5] pertigaan_S.Parman-RadenWijaya-KusumaBangsa
[6] belokan_KusumaBangsa1
[7] belokan_KusumaBangsa2
[8] pertigaan_KusumaBangsa-Pemuda
[9] pertigaan_Pemuda-GajahMada
[10] pertigaan_Pemuda-Airlangga-HayamWuruk
[11] perempatan_HayamWuruk-Kartini-Brawijaya-Niaga
[12] perempatan_AwangAwang
[13] pertigaan_Leminggir
[14] perempatan_JembatanTuri
[15] pertigaan_JembatanTuri
```

```
Start: 3
End: 15
Expanded Node: [3, 1, 2, 4, 5, 6, 7, 12, 10, 9, 8] -> X [NOT FOUND]
Tidak ada rute yang menghubungkan kedua titik.
```



Gambar 2.10, 2.11. Contoh graf daerah sekitar Mojosari, Mojokerto dan kasus pencarian jalur tidak ditemukan

```
Start: 3
End: 11
Expanded Node: [3, 12, 10] -> 11 [FOUND]

3 - belokan_A.Yani [0.000000 km]
12 - perempatan_AwangAwang [3.229305 km]
10 - pertigaan_Pemuda-Airlangga-HayamWuruk [1.731642 km]
11 - perempatan_HayamWuruk-Kartini-Brawijaya-Niaga [0.622982 km]

Jarak total: 5.583929 km
Path: [3] belokan_A.Yani -> [12] perempatan_AwangAwang -> [10] pertigaan_Pemuda-Airlangga-HayamWuruk -> [11] perempatan_HayamWu
ruk-Kartini-Brawijaya-Niaga
```



Gambar 2.12. Contoh graf daerah Mojosari, Mojokerto dan kasus pencarian jalur ditemukan

Bab III
Kesimpulan

Pada akhir pembuatan program dan pengujian berdasarkan 4 sampel tipe file eksternal, algoritma A* berhasil diimplementasikan pada program dan kesemuanya menghasilkan output yang benar. Kami menarik kesimpulan bahwa algoritma A* ini akan hampir selalu menemukan solusi paling optimum, meskipun masih terdapat kekurangan yaitu kompleksitasnya yang tidak begitu mangkus.

Repository lengkap dapat diakses pada : [naufalizza/tucil3stima: Tucil 3 Stima: (github.com)]

Centang (V) jika ya

1	Program dapat menerima input graf	V
2	Program dapat menghitung lintasan terpendek	V
3	Program dapat menampilkan lintasan terpendek serta jaraknya	V
4	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta	[cek keterangan]

Keterangan poin 4 : Map bisa ditampilkan dengan openstreetmap, tetapi input melalui file eksternal, bukan melalui klik pada peta.