

LAPORAN TUGAS KECIL 3 MATA KULIAH IF2211 STRATEGI ALGORITMA

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Disusun untuk Memenuhi Tugas Kecil 3 Mata Kuliah IF2211 Strategi Algoritma

Nama	NIM
Rais Vaza Man Tazakka	(13519060)
Prana Gusriana	(13519195)



**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021**

DAFTAR ISI

DAFTAR ISI	2
BAB I : DESKRIPSI MASALAH	3
I. Deskripsi Masalah	3
BAB II : PEMBAHASAN	5
I. Strategi Penyelesaian	5
BAB III : IMPLEMENTASI PROGRAM	6
BAB IV : EKSPERIMEN	21
I. File Input	21
1. peta-itb.txt	21
2. peta-alun-alun.txt	22
3. peta-buahbatu.txt	23
4. peta-sekitar.txt	23
II. Hasil Running Program	24
1. peta-itb.txt	24
2. peta-alun-alun.txt	27
3. peta-buahbatu.txt	29
4. peta-sekitar.txt	31
BAB V: KESIMPULAN, SARAN, REFLEKSI, DAN KOMENTAR	35
I. Kesimpulan	35
II. Saran	35
III. Refleksi dan Komentar	35
DAFTAR PUSTAKA	36

BAB I

DESKRIPSI MASALAH

I. Deskripsi Masalah

Algoritma 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 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.



Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar kampus ITB). Sisi diperoleh dari jalan antar dua simpul dan bobot sisi adalah jarak Euclidean. Berdasarkan graf yang dibentuk, lalu program A* menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya. Lintasan terpendek dapat ditampilkan pada peta/graf. 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.

Berkas yang dikumpulkan: Laporan berisi kode program, peta/graf input, *screenshot* peta yang memperlihatkan lintasan terpendek untuk sepasang simpul, alamat tempat kode sumber program diletakkan jika perlu dieksekusi oleh asisten. Tampilkan hasil untuk beberapa lintasan terpendek.

Peta jalan yang digunakan sebagai kasus uji adalah:

1. Peta jalan sekitar kampus ITB/Dago
2. Peta jalan sekitar Alun-alun Bandung
3. Peta jalan sekitar Buahbatu
4. Peta jalan sebuah kawasan di kotamu

BAB II

PEMBAHASAN

I. Strategi Penyelesaian

Untuk dapat mencari lintasan terpendek dari sebuah input file graf pada peta kami menggunakan algoritma A* (Astar). Algoritma A* yang kami buat hampir mirip seperti algoritma BFS (Breadth First Search) karena pencarian lintasan terpendek dilakukan dengan membangkitkan simpul yang bertetangga dengan simpul yang sedang diperiksa tetapi pada algoritma A* urutan pembangkitan simpulnya diurutkan berdasarkan nilai $F(n)$. $F(n)$ merupakan heuristic evaluation function. Pada algoritma A* $F(n)$ merupakan estimasi total biaya dari suatu lintasan yang melalui n ke tujuan, nilai dari $F(n)$ adalah $F(n) = g(n) + h(n)$. $g(n)$ merupakan biaya seberapa jauh untuk mencapai simpul n dari simpul awal serta $h(n)$ merupakan estimasi biaya dari simpul n ke simpul goal. Ide dari algoritma A* ini adalah menghindari untuk membangkitkan lintasan yang costnya mahal. Pada penggeraan tugas kecil 3 ini, nilai $g(n)$ merupakan total jarak yang telah ditempuh dari simpul awal ke simpul n dan nilai $h(n)$ merupakan jarak garis lurus dari simpul n ke simpul goal.

Untuk dapat menyelesaikan persoalan sesuai dengan spesifikasi yang diminta, program yang kami buat dapat meminta nama file input dari user, memvisualisasikan peta berdasarkan file input, mencari lintasan terpendek dari simpul awal ke simpul goal, dan juga program dapat memvisualisasikan lintasan terpendek tersebut pada peta. Untuk detail seperti requirement, cara menggunakan program, format file input dapat dilihat di README.md pada [github](#) (dapat diakses pada Rabu, 7 April 2021 pukul 13.00).

1	Program dapat menerima input graf	✓
2	Program dapat menghitung lintasan terpendek	✓
3	Program dapat menampilkan lintasan terpendek serta jaraknya	✓
4	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta	

BAB III

IMPLEMENTASI PROGRAM

Dalam penggerjaan tugas kecil 3 untuk mencari lintasan terpendek dengan menggunakan algoritma A* ini saya menggunakan python dan jupyter notebook. Kami mendekomposisi persoalannya menjadi empat bagian, yaitu untuk memproses file (persoalan membaca dan mengolah input file), kelas graf (menyimpan informasi representasi graf dari file input), kelas *priority queue* (untuk membantu program pencarian lintasan terpendek dengan cara mengurutkan simpul yang akan diekspan sesuai dengan nilai F(n)), dan *main program*. Semua *source code* terdapat dalam satu folder yang sama yaitu folder src. *Source code* untuk memproses file yaitu FileProcessing.py, *source code* untuk kelas graf adalah Graf.py, *source code* untuk kelas *priority queue* adalah PriorityQueue.py, dan *source code* untuk main programnya yaitu MainProgram.ipynb. Penjelasan cara kerja fungsi dapat langsung dilihat pada source code. *Source code*, *test case*, cara penggunaan program, *requirement*, dan juga dokumentasi dapat diakses di [github](#) (dapat diakses mulai dari Rabu, 7 April 2021 pukul 13.00) berikut ini. Berikut adalah source code yang sudah kami kerjakan.

1. FileProcessing.py

```
# Untuk membaca file input yang terdapat di folder test

def readInputFromFile(namaFile, arrayv, graf):

    file = open("../test/" + namaFile, "r")

    file_input = file.readlines()

    nvertex = int(file_input[0])

    graftemp = [[0 for j in range(nvertex)] for i in range(nvertex)]

    i = 1

    while(i < nvertex+1):

        j = 0

        nama_simpul = ""
```

```

koordinat = []

tempstr = file_input[i].split()

while (j < len(tempstr)):

    if(j==0):

        nama_simpul += tempstr[j]

    else:

        koordinat += [float(tempstr[j])]

    j += 1

arrayv += [[nama_simpul, koordinat]]

i += 1

while(i < 2*nvertex+1):

    tempstr = file_input[i].split()

    for j in range(len(tempstr)):

        graftemp[i-nvertex-1][j] = float(tempstr[j])

    i += 1

file.close()

graf += graftemp

```

2. PriorityQueue.py

"""

Class PriorityQueue: digunakan sebagai queue di class graf untuk membantu pencarian lintasan terpendek dengan menggunakan algoritma A*

-> Mempunyai atribut:

- queue: array berukuran maxEl untuk menyimpan elemen queue yang berupa list of [namaSimpul, [latitude, longitude], fn] (lintasan)

- neff: jumlah elemen yang telah tersimpan pada queue

- **MaxEl** : Max element dari queue

-> Mempunyai method:

- **__init__** : konstruktor class PriorityQueue user defined maxEl

- **push** : untuk menambahkan element pada queue sesuai dengan nilai fn

- **pop** : menghapus elemen paling awal lalu mengembalikan nilai elemen paling awal tersebut

- **getNeff** : mendapatkan neff

- **getQueue** : mendapatkan queue

- **getMaxEl** : mendapatkan maxEl

- **printPriorityQueue**: untuk menampilkan neff, maxEL, dan seluruh elemen queue

""""

```
class PriorityQueue:
```

```
    def __init__(self, maxEl):
```

```
        self.queue = [0 for i in range(maxEl)]
```

```
        self.neff = 0
```

```
        self.maxEl = maxEl
```

```
    def push(self, element):
```

```
        idx = self.neff
```

```
        while(idx > 0):
```

```
            if(element[len(element)-1][2] < self.queue[idx-1][len(self.queue[idx-1])-1][2]):
```

```
                self.queue[idx] = self.queue[idx-1]
```

```
                idx -= 1
```

```
            else:
```

```
                break
```

```
        self.queue[idx] = element
```

```
        self.neff += 1
```

```
    def pop(self):
```

```

temp = self.queue[0]
for i in range(self.neff-1):
    self.queue[i] = self.queue[i+1]
self.neff -= 1
return temp

def getNeff(self):
    return self.neff

def getQueue(self):
    return self.queue

def getMaxEl(self):
    return self.maxEl

def printPriorityQueue(self):
    print("Neff:", self.neff)
    print("MaxEL:", self.maxEl)
    for i in range(self.neff):
        print(self.queue[i])

```

3. Graf.py

```

import PriorityQueue
import math

```

```
"""
```

Class Graf: digunakan untuk merepresentasikan graf dari peta, dengan simpul berupa suatu koordinat pada peta dengan format NamaSimpul latitude longitude

-> Mempunyai atribut:

- graf : matriks ketetanggaan berbobot dengan bobot berupa jarak antara simpul yang bertetangga

- hn : metriks dengan elemen berupa jarak dari suatu simpul ke simpul lainnya, diolah oleh program
- vertex : array of vertex yang terdapat pada graf
- vertexidx: dictionary dengan key berupa namasimpul dan value berupa indeks pada matriks ketetanggaan yang relate dengan nama simpul
- visited : dictionary of boolean untuk menyimpan informasi apakah suatu simpul telah dikunjungi atau belum
- solutionPath: list of path yang merupakan shortest path dari simpul awal ke simpul goal
- queue: priority queue dengan elemen merupakan list of path

-> Mempunyai method:

- initializeQueue: Menginisialisasi objek priority queue, dipanggil pada method Astar untuk menginisialisasi queue awal
- addVertex: Menambahkan vertex dengan indeks sesuai pada graf ketetanggaan
- haversineFormula: Untuk menghitung jarak antar dua koordinat di peta
- getHnMatriks: Method yang digunakan untuk membentuk atribut hn
- addGraf: Menambahkan graf ketetanggaan dengan parameter matriks ketetanggaan dari inputfile (user harus menginput matriks ketetanggaannya sendiri)
- buildGrafBool: Mengecek dan mengubah input dari graf jika input dari graf merupakan matriks adjacency boolean, jika telah dalam bentuk jarak tidak akan diubah
- total_jarak: Menghitung total jarak dari suatu path (jarak dari simpul awal ke simpul n)
- Astar: Algoritma untuk mencari shortest path
- findShortestPath: Method yang dipanggil user untuk membentuk matriks hn (getHnMatriks), mencari jalur terdekat(Astar), dan menampilkan solusi(printSolusi)
- getSolution: Mengembalikan list of path
- isVertex: Mengecek apakah suatu input nama simpul merupakan simpul dalam graf
- printSolusi: Mencetak solusi dalam bentuk text berupa path dan total jarak yang dilaluinya
- clearGraf: Mengosongkan graf agar bisa digunakan berulangkali

-> CATATAN: Untuk mencari jalur terpendek gunakan langsung method findShortestPath, JANGAN memanggil algoritma Astar langsung

"""

```

class Graf:
    def __init__(self):
        self.graf = [] # Matriks ketetanggaan dengan bobot merupakan jarak antar simpul
        self.hn = [] # Matriks dengan elemen berupa jarak dari suatu simpul ke simpul lainnya
        self.vertex = [] # Array dari kumpulan simpul dengan format [namasimpul, [latitude, longitude]]
        self.vertexidx = {} # Dictionary untuk mendapatkan indeks simpul dengan key berupa namasimpul dan value integer
        self.visited = {} # Dictionary untuk mengecek apakah suatu simpul telah dikunjungi atau belum dengan key berupa namasimpul dan value boolean
        self.solutionPath = [] # Untuk menyimpan solusi, berupa list of path
        self.queue = [] # Queue untuk membantu penarian solusi dengan menggunakan algoritma A*
# Inisialisasi queue sebagai objek PriorityQueue dengan maxEl jumlah simpul^2
    def initializeQueue(self):
        self.queue = PriorityQueue.PriorityQueue(len(self.vertex) ** 2)

# Menambah simpul dengan format vertex adalah [namasimpul, [latitude, longitude]]
# Menambahkan indeks pada matriks sesuai dengan urutan
# Dipanggil ketika membaca input graf secara iteratif sehingga indeksnya dapat ditentukan
# Contoh terdapat 8 simpul maka indeks mulai dari 0 sampai 7
    def addVertex(self, vertex, idx):
        self.vertex += [vertex]
        self.vertexidx[vertex[0]] = idx

# Mencari jarak antar dua koordinat
    def haversineFormula(self, Lat1, Long1, Lat2, Long2):

```

```

dLat = ((Lat2-Lat1)*math.pi)/180
dLong = ((Long2-Long1)*math.pi)/180
a = ((math.sin(dLat/2)) ** 2) + math.cos(Lat1 * math.pi/180) * math.cos(Lat2
* math.pi /180) * ((math.sin(dLong/2)) ** 2)
c = 2 * math.asin(math.sqrt(a))
return 6371000 * c

```

Membuat matriks hn dimana isinya merupakan jarak dari suatu simpul ke simpul lainnya

```

def getHnMatrix(self):
    self.hn = [[0 for j in range(len(self.vertex))] for i in range(len(self.vertex))]
    for vertex1 in self.vertex:
        for vertex2 in self.vertex:
            if(vertex1[0] == vertex2[0]):
                self.hn[self.vertexidx[vertex1[0]]][self.vertexidx[vertex2[0]]] = 0
            else:
                self.hn[self.vertexidx[vertex1[0]]][self.vertexidx[vertex2[0]]] =
                self.haversineFormula(vertex1[1][0], vertex1[1][1], vertex2[1][0], vertex2[1][1])

```

Menambahkan representasi graf adjacency matriks yang telah diolah dari input file

```

def addGraf(self, graf):
    self.graf = graf

```

Method untuk menghitung jarak total dari current path, dari simpul awal ke simpul n

```

def total_jarak(self, path):
    if(len(path) <= 1):
        return 0

```

```

else:
    total = 0
    for i in range(len(path)-1):
        total += self.graf[self.vertexidx[path[i][0]]][self.vertexidx[path[i+1][0]]]
    return total

# Algoritma A* untuk mencari jalur terpendek
# Secara umum algoritmanya seperti BFS namun queue yang digunakan adalah
Priority Queue yang sudah disesuaikan

def Astar(self, Vfirst, Vgoal):
    self.initializeQueue() # Inisialisasi queue
    self.queue.push([self.vertex[self.vertexidx[Vfirst]]+[0]]) # add list of vertex ke
queue dengan format [namasimpul, [latitude, longitude], f(n)]
    for i in self.vertex:
        self.visited[i[0]] = False # Inisialisasi semua simpul belum dikunjungi
    while(self.queue.getNeff()>0): # Ketika queue belum kosong
        path = self.queue.pop() # pop queue untuk mendapatkan path yang
akan diperiksa
        total_jarak = self.total_jarak(path) # Total jarak dari simpul awal ke
simpul yang sedang diperiksa
        Vcurr = path[-1] # Mendapatkan simpul yang akan diperiksa dengan
format [namasimpul, [latitude, longitude], f(n)]
        self.visited[Vcurr[0]] = True # Tandai Vcurr telah dikunjungi
        if (Vcurr[0] == Vgoal): # Jika simpul solusi diperiksa maka masukkan
kedalam solution path dan pencarian dihentikan
            self.solutionPath.append(path)
        return
        # Tambahkan simpul yang bertetangga ke queue
        for adjacent in range(len(self.graf[self.vertexidx[Vcurr[0]]])):
            # Jika bobot Vcurr dan adjacent tidak sama dengan 0 serta
simpul yang bertetangga belum dikunjungi

```

```

        if (self.graf[self.vertexidx[Vcurr[0]]][adjacent] != 0 and
self.visited[self.vertex[adjacent][0]] == False):
            # Tambahkan kedalam queue
            newPath = list(path)
            newPath.append(self.vertex[adjacent] + [total_jarak +
self.graf[self.vertexidx[Vcurr[0]]][adjacent] + self.hn[adjacent][self.vertexidx[Vgoal]]])
            self.queue.push(newPath)

# Untuk mencari jalur terpendek, user harus memanggil method ini
def findShortestPath(self, Vfirst, Vgoal):
    self.buildGrafBool()
    self.getHnMatrix()
    self.solutionPath = []
    self.Astar(Vfirst, Vgoal)
    self.printSolusi()

# Method untuk mendapatkan solusi
def getSolution(self):
    return self.solutionPath

# Method untuk mengecek suatu input string merupakan simpul dari graf atau
bukan
def isVertex(self, vertex):
    for v in self.vertex:
        if(v[0] == vertex):
            return True
    return False

# Untuk menampilkan shortest path yang berbentuk text
def printSolusi(self):
    if(len(self.solutionPath) == 0):

```

```

        print("Tidak ada solusi")

    else:
        strPrint = "Shortest path dari " + self.solutionPath[0][0][0] + " ke " +
        self.solutionPath[0][len(self.solutionPath[0])-1][0] + " adalah\n"
        for vertex in self.solutionPath[0]:
            strPrint += vertex[0]
            if(vertex[0] != self.solutionPath[0][len(self.solutionPath[0])-1][0]):
                strPrint += " -> "
            else:
                strPrint += "\nDengan total jarak adalah " +
        str(vertex[2]) + " meter"
        print(strPrint)

# Mengosongkan graf
def clearGraf(self):
    self.graf = []
    self.hn = []
    self.vertex = []
    self.vertexidx = {}

# Mengecek dan mengubah graf yang berupa matriks adjacency boolean
# Jika input bobot antar simpul telah dalam bentuk jarak maka tidak akan diubah
def buildGrafBool(self):
    cekGraf = {}
    for i in self.graf:
        for j in i:
            cekGraf[j] = j
    for key in cekGraf:
        if(key != 0 and key != 1):
            return

```

```

for vertex1 in self.vertex:
    for vertex2 in self.vertex:

if(self.graf[self.vertexidx[vertex1[0]]][self.vertexidx[vertex2[0]]] == 0):

    self.graf[self.vertexidx[vertex1[0]]][self.vertexidx[vertex2[0]]] = 0

    else:

        self.graf[self.vertexidx[vertex1[0]]][self.vertexidx[vertex2[0]]] =
        self.haversineFormula(vertex1[1][0], vertex1[1][1], vertex2[1][0], vertex2[1][1])

```

4. MainProgram.ipynb

```

# Cell 0, untuk mengimport library

# CATATAN: Program dijalankan secara berurutan dari cell 0 sampai cell 4

#     Program dapat mencari jalur terpendek dari berbagai simpul dari suatu input file
# secara terus menerus dengan melakukan running pada cell 3

#     - Jika ingin melihat visualisasinya silahkan lakukan running pada cell 4, jika
# tidak hanya akan menampilkan shortest path berbentuk text

#     Jika ingin mengganti file input bisa melakukan running pada cell 1 kemudian
# tampilkan visualisasi awalnya pada cell 2

# Jika terdapat kesalahan yang diluar dugaan, bisa melakukan restart kernel dan jalankan
# kembali program dari awal

import PriorityQueue

import Graf

import FileProcessing

import math

import folium

import os

```

```

graf_peta = Graf.Graf() #Terdapat 1 inisialisasi kelas graf, isi dari graf dikelola sehingga
dapat digunakan berulang kali

# Cell 1, untuk membaca file input dari user

# File berupa txt dengan isi

# Baris pertama merupakan sebuah integer N yang merepresentasikan jumlah simpul

# Baris selanjutnya merupakan N buah simpul dengan format: NamaSimpul(nama simpul
# tanpa spasi) <spasi> Latitude <spasi> Longitude

# Selanjutnya merupakan graf ketetanggaan berbobot yang merepresentasikan hubungan
# antar simpul dalam graf dengan bobot merupakan jarak antar simpul tersebut atau bisa
# juga boolean

# Jika tidak terdapat hubungan diisi dengan 0, dan dengan simpul lain yang berhubungan
# bisa diisi jarak kedua simpul tersebut atau 1

array_of_vertex = []

adjacency_matriks_graf = []

N = input("Masukkan nama file: ")

while (not(os.path.isfile("../test/" + N))):  

    print("File tidak ditemukan silahkan masukkan kembali input nama file,")

    N = input("Masukkan nama file: ")

FileProcessing.readInputFromFile(N, array_of_vertex, adjacency_matriks_graf)

# Cell 2, untuk membangun graf dari file input serta menampilkan visualisasi graf awal

# Klik pada penanda di peta untuk mengetahui nama simpul

visualisasi_map = folium.Map(location=array_of_vertex[0][1], zoom_start=17)

```

```

idx = 0

graf_peta.clearGraf()

for vertex in array_of_vertex:

    graf_peta.addVertex(vertex, idx)

    folium.Marker(location=vertex[1], popup='<h3 style="color:green;">' + vertex[0] +
    '</h3>', tooltip='<strong>Click here to see nama simpul</strong>',
    icon=folium.Icon(color='red', Icon='none')).add_to(visualisasi_map)

    idx += 1

graf_peta.addGraf(adjacency_matriks_graf)

visualisasi_map

# Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya
dengan algoritma A*

# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta

# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan
menggunakan text

# Visualisasi jalur terdapat pada cell 4

print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^")

simpul_awal = str(input("Masukkan simpul awal: "))

while(not(graf_peta.isVertex(simpul_awal))):

    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input
simpul awal")

    simpul_awal = str(input("Masukkan simpul awal: "))

simpul_goal = str(input("Masukkan simpul tujuan: "))

```

```

while(not(graf_peta.isVertex(simpul_goal))):  

    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input  

simpul tujuan")  

simpul_goal = str(input("Masukkan simpul tujuan: "))  

graf_peta.findShortestPath(simpul_awal, simpul_goal)  

# Cell 4, untuk memvisualisasikan jalur terpendek pada map  

# Klik pada jalur untuk melihat jaraknya  

coorSol = []  

array_solution = graf_peta.getSolution()  

visualisasi_shortest_path = folium.Map(location=array_of_vertex[0][1], zoom_start=17)  

for vertex in array_of_vertex:  

    folium.Marker(location=vertex[1], popup='<h3 style="color:green;">' + vertex[0] +  

'</h3>', tooltip='<strong>Click here to see nama simpul</strong>',  

icon=folium.Icon(color='red', icon='none')).add_to(visualisasi_shortest_path)  

if(len(array_solution) != 0):  

    for vertex in array_solution[0]:  

        coorSol += [vertex[1]]  

strGrp = simpul_awal + "-" + simpul_goal  

f1 = folium.FeatureGroup(strGrp)  

solutionpath = folium.vector_layers.PolyLine(coorSol,  

popup='<b>' + str(array_solution[0][len(array_solution[0])-1][2]) + ' meter</b>',  

tooltip='shortest path', color='red', weight=10).add_to(f1)

```

```
f1.add_to(visualisasi_shortest_path)  
folium.LayerControl().add_to(visualisasi_shortest_path)  
  
visualisasi_shortest_path
```

BAB IV

EKSPERIMEN

I. File Input

Kami membuat 4 data uji sesuai yang tertulis di spesifikasi tugas kecil 3, yaitu peta jalan sekitar kampus ITB/Dago (peta-itb.txt), peta sekitar alun-alun Bandung (peta-alun-alun.txt), peta jalan sekitar Buahbatu (peta-buahbatu.txt), dan peta jalan sebuah kawasan di kotamu (peta-sekitar.txt).

1. peta-itb.txt

28

Gerbang_depan -6.893229 107.610454
Persimpangan_1A -6.892651 107.610431
Persimpangan_AB -6.891932 107.610391
Persimpangan_BB1 -6.891946 107.609772
Persimpangan_BB2 -6.891891 107.610954
Campus_Center -6.891014 107.610395
Plawid -6.889924 107.610371
Persimpangan_B21 -6.891040 107.609711
Persimpangan_B22 -6.891041 107.611046
Persimpangan_C2 -6.891062 107.608698
Persimpangan_G2 -6.891013 107.611574
GKUB -6.890370 107.608823
GKUT -6.890247 107.611884
Gerbang_sipil -6.893703 107.608783
Gerbang_SR -6.893547 107.611942
Persimpangan_1CD -6.892669 107.608770
Persimpangan_DD1 -6.892441 107.608299
Persimpangan_DD2 -6.891081 107.608221
Persimpangan_1SR -6.892418 107.611910
Persimpangan_111 -6.891988 107.612161
Persimpangan_12 -6.890969 107.612083
Persimpangan_DE4 -6.890012 107.609056
Persimpangan_G4 -6.889926 107.611578
Labtek5 -6.890565 107.609725
Labtek6 -6.890158 107.609775
Labtek8 -6.890571 107.611008

2. peta-alun-alun.txt

10 Simpang_Lima -6.922378975960281 107.6176678307514
Jl_Lengkong -6.921793191149624 107.61193326823145
JL_Sudirman -6.920818691162017 107.60408784213622
Jl_AstanaAnyar -6.920085791953785 107.59835126827026
Jl_Pasirkoja -6.926944778776833 107.60004105984538
Jl_Pungkur1 -6.927301571170082 107.60365131295659
Jl_Pungkur2 -6.931266999932228 107.61226072110998
Jl_Pungkur3 -6.932502080371825 107.61484010807965
Jl_Karapitan -6.928773321034194 107.61644914900161
Jl_LengkongBesar -6.927281813095968 107.61315282099183

0 1 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 0 0 1
0 1 0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0 0 0
0 0 0 0 0 1 0 1 0 1
0 0 0 0 0 1 0 1 0 0
1 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 1 0 1 0

3. peta-buahbatu.txt

10

Simpang_Terusan_BuahBatu -6.94802784162572 107.63341750782924
Simpang_BKR -6.93697379595095 107.6227191373766
Sakura_Derma -6.9318958321255835 107.61800071546168
Simpang_Horison -6.934905002481361 107.62482573683579
Simpang_PelajarPejuang -6.932484344657527 107.62567468156436
Simpang_GatotSubroto -6.924560381957079 107.62769706720903
Simpang_Turangga -6.927633447267659 107.63593383253776
MartaNegara -6.934380495598323 107.63385780288841
Simpang_Kircon -6.9454033629317244 107.64189475068316
Simpang_IbrahimAdjie -6.931747284811854 107.64308862313439

0 1 0 0 0 0 0 0 1 0
1 0 1 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0
0 0 0 1 0 1 0 1 0 0
0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 1 0 1 0 1
0 0 0 0 1 0 1 0 0 0
1 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0 1 0

4. peta-sekitar.txt

19

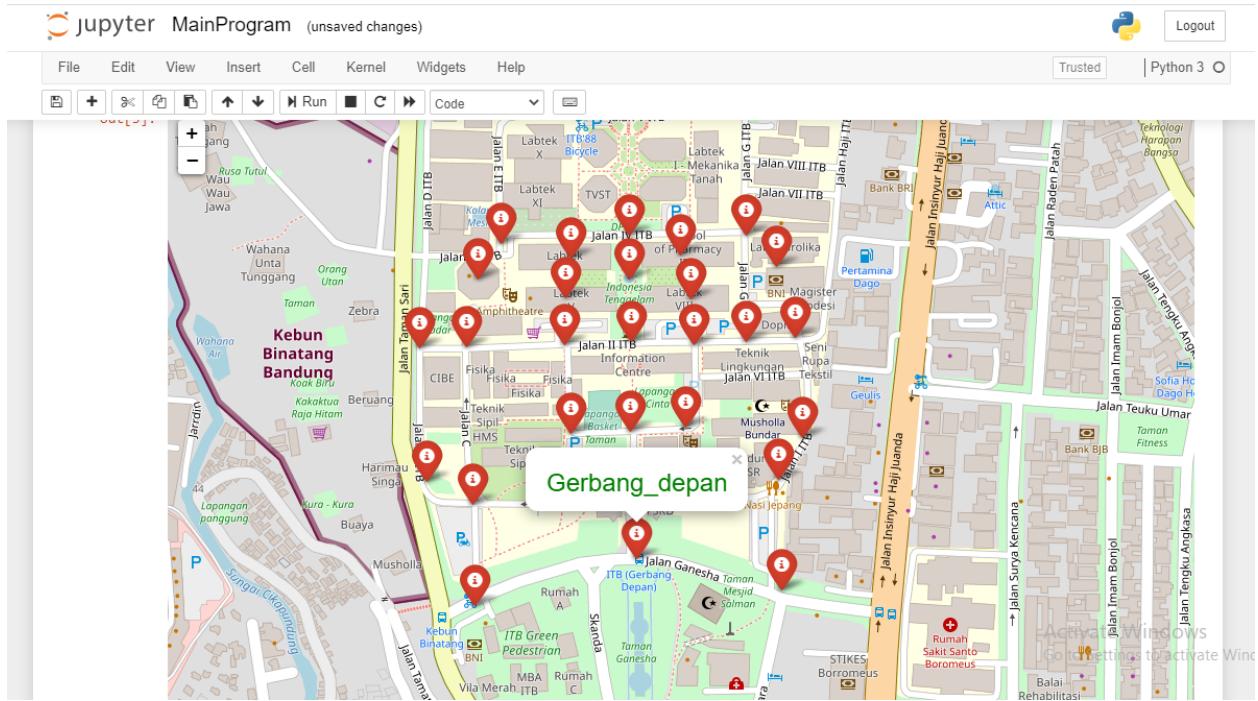
SMANOP -6.919471 107.598340
JA -6.920112 107.598348
PA -6.923271 107.598662
JC -6.919497 107.592915
CL -6.920743 107.592722

PJ -6.921961 107.586017
PS -6.925114 107.599285
L14 -6.920692 107.591214
JJ -6.918810 107.586511
AP -6.930164 107.600777
SN -6.925155 107.596102
L1P -6.922451 107.591190
PK -6.929896 107.597336
PKJ -6.926765 107.598225
KP -6.936987 107.595009
FCL -6.929265 107.587368
SPK -6.926495 107.596015
TPJ -6.925868 107.588277
Perlmaan -6.926795 107.585533
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 0 1 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 1 0 0 0 0 0 0 1 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 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0

II. Hasil Running Program

1. peta-itb.txt

Berikut adalah screenshot dari visualisasi peta berdasarkan file input



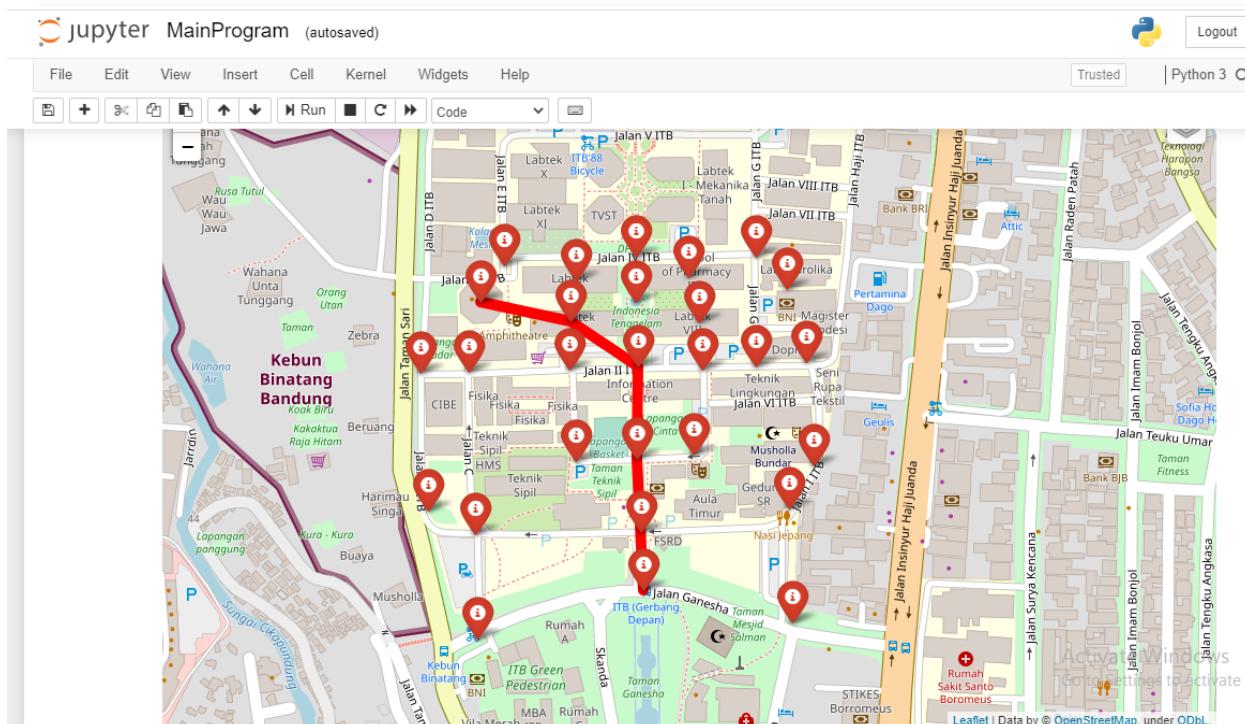
Berikut adalah pencarian lintasan terdekat dari Gerbang_depan ke GKUB

jupyter MainProgram (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 Logout

```
In [4]: # Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^ ^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal)):
    print("tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
    simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal)):
    print("tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
    simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^ ^
Masukkan simpul awal: gerbang
Tidak terdapat simpul gerbang, silahkan masukkan kembali input simpul awal
Masukkan simpul awal: Gerbang_depan
Masukkan simpul tujuan: GKUB
Shortest path dari Gerbang_depan ke GKUB adalah
Gerbang_depan -> Persimpangan_1A -> Persimpangan_AB -> Campus_Center -> Labtek5 -> GKUB
Dengan total jarak adalah 437.6128119205004 meter
```



Berikut adalah pencarian lintasan terdekat dari GKUB ke Gerbang_sipil

```
In [6]: # Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal))):
    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal))):
    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)
```

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^

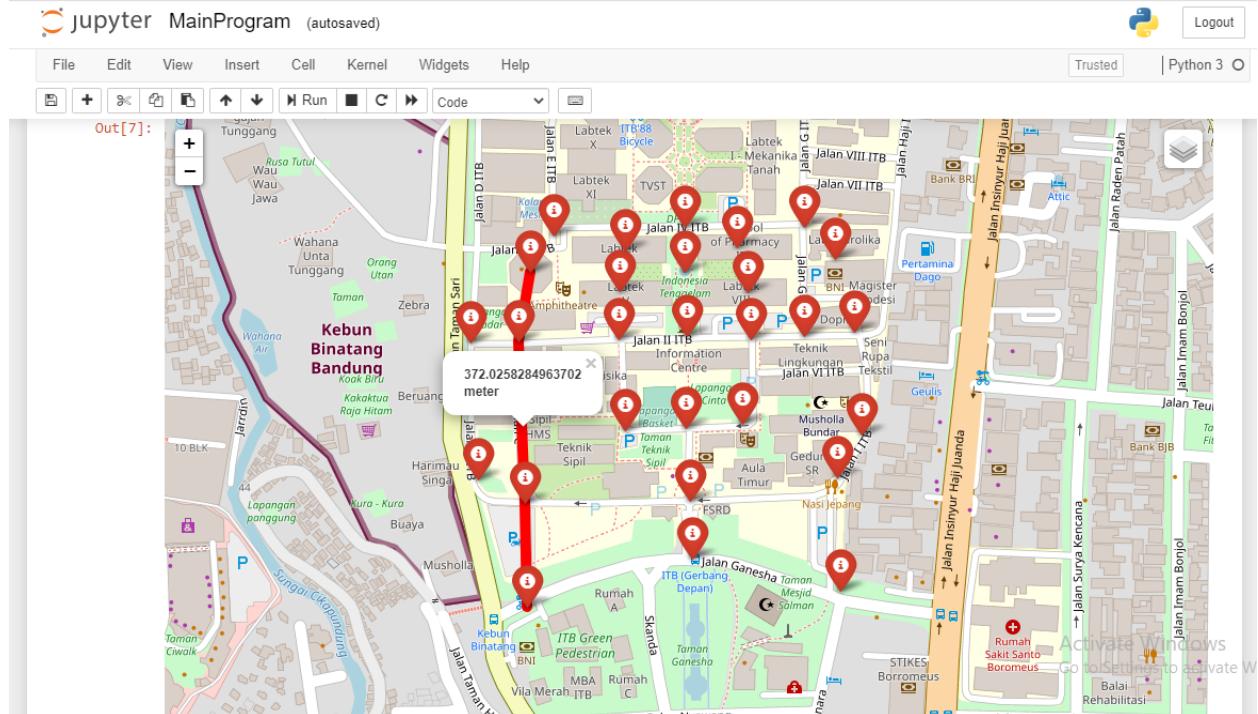
Masukkan simpul awal: GKUB

Masukkan simpul tujuan: Gerbang_sipil

Shortest path dari GKUB ke Gerbang_sipil adalah

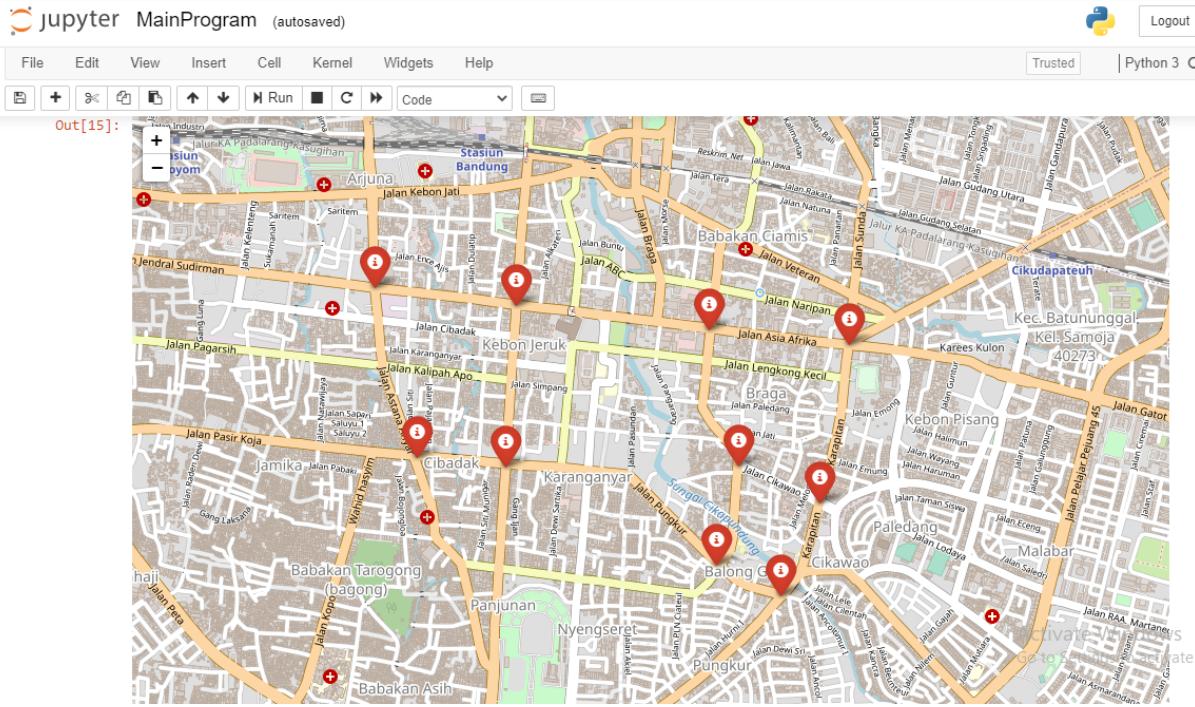
GKUB -> Persimpangan_C2 -> Persimpangan_1CD -> Gerbang_sipil

Dengan total jarak adalah 372.0258284963702 meter



2. peta-alun-alun.txt

Berikut adalah screenshot dari visualisasi peta berdasarkan file input



Berikut adalah hasil pencarian lintasan terpendek dari Simpang_Lima ke Jl_Pasirkoja

jupyter MainProgram (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [16]: # Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal)):
    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
    simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal)):
    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
    simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^
Masukkan simpul awal: Simpang_lima
Tidak terdapat simpul Simpang_lima, silahkan masukkan kembali input simpul awal
Masukkan simpul awal: Simpang_Lima
Masukkan simpul tujuan: Jl_Pasirkoja
Shortest path dari Simpang_Lima ke Jl_Pasirkoja adalah
Simpang_Lima -> Jl_Lengkong -> Jl_Sudirman -> Jl_Pungkur1 -> Jl_Pasirkoja
Dengan total jarak adalah 2632.069402533828 meter
```

In [13]: # Cell 4, untuk memvisualisasikan jalur terpendek pada map
Klik pada jalur untuk melihat jaraknya

Activate Windows Go to Settings to activate Wi

jupyter MainProgram (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

3. peta-buahbatu.txt

Berikut adalah screenshot dari visualisasi peta berdasarkan file input

jupyter MainProgram (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Masukkan nama file: peta-buahbatu.txt

```
In [19]: # Cell 2, untuk membangun graf dari file input serta menampilkan visualisasi graf awal
# Klik pada penanda di peta untuk mengetahui nama simpul
visualisasi_map = folium.Map(location=array_of_vertex[0][1], zoom_start=17)
idx = 0
graf_peta.clearGraf()
for vertex in array_of_vertex:
    graf_peta.addVertex(vertex, idx)
    folium.Marker(location=vertex[1], popup='<h3 style="color:green;">' + vertex[0] + '</h3>', tooltip='<strong>Click here to see details</strong>')
    idx += 1
graf_peta.addDraf(adjacency_matriks_graf)
visualisasi_map
```

Out[19]:

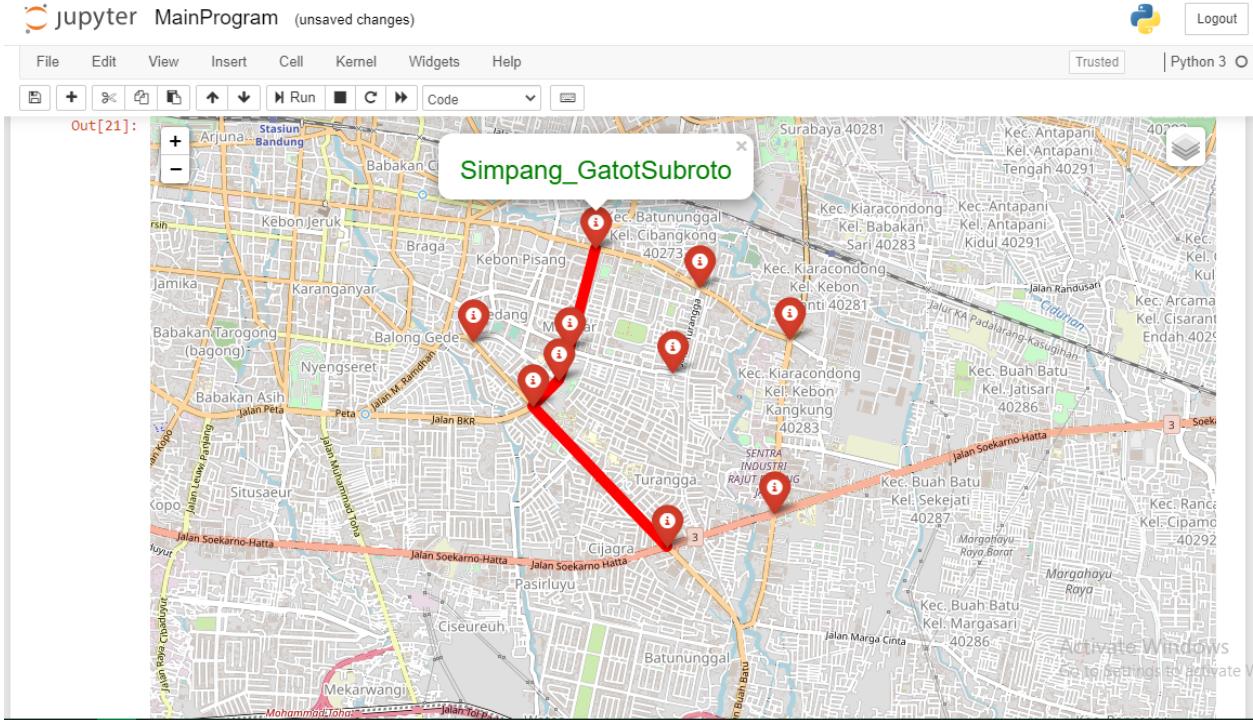
Berikut adalah hasil pencarian lintasan terdekat dari Simpang_Terusan_BuahBatu ke Simpang_GatotSubroto

jupyter MainProgram (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 Logout

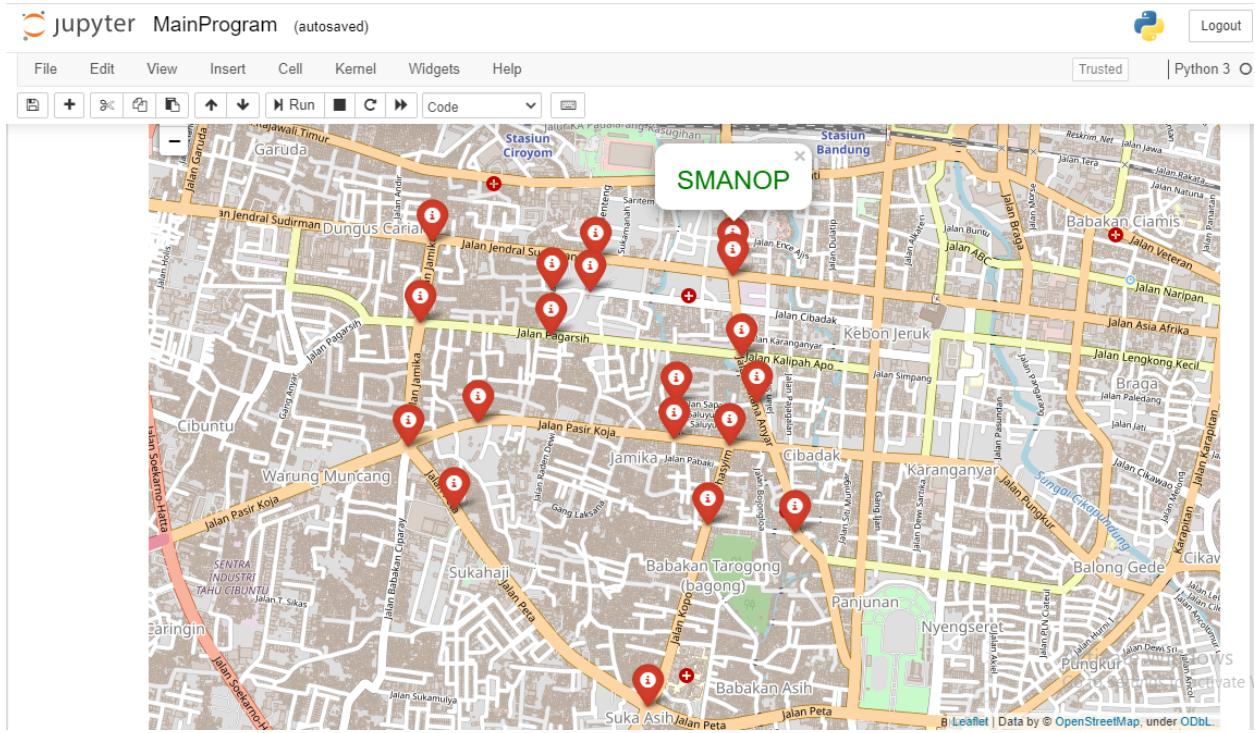
```
In [20]: # Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal)):
    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
    simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal)):
    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
    simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^
Masukkan simpul awal: Simpang_TerusanBuahBatu
Tidak terdapat simpul Simpang_TerusanBuahBatu, silahkan masukkan kembali input simpul awal
Masukkan simpul awal: Simpang_Terusan_BuahBatu
Masukkan simpul tujuan: Simpang_GatotSubroto
Shortest path dari Simpang_Terusan_BuahBatu ke Simpang_GatotSubroto adalah
Simpang_Terusan_BuahBatu -> Simpang_BKR -> Simpang_Horison -> Simpang_PelajarPejuang -> Simpang_GatotSubroto
Dengan total jarak adalah 3225.539213815977 meter
```



4. peta-sekitar.txt

Berikut adalah screenshot dari visualisasi peta berdasarkan file input



Berikut adalah hasil pencarian lintasan terdekat dari SMANOP ke FCL

C localhost:8888/notebooks/MainProgram.ipynb

jupyter MainProgram (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

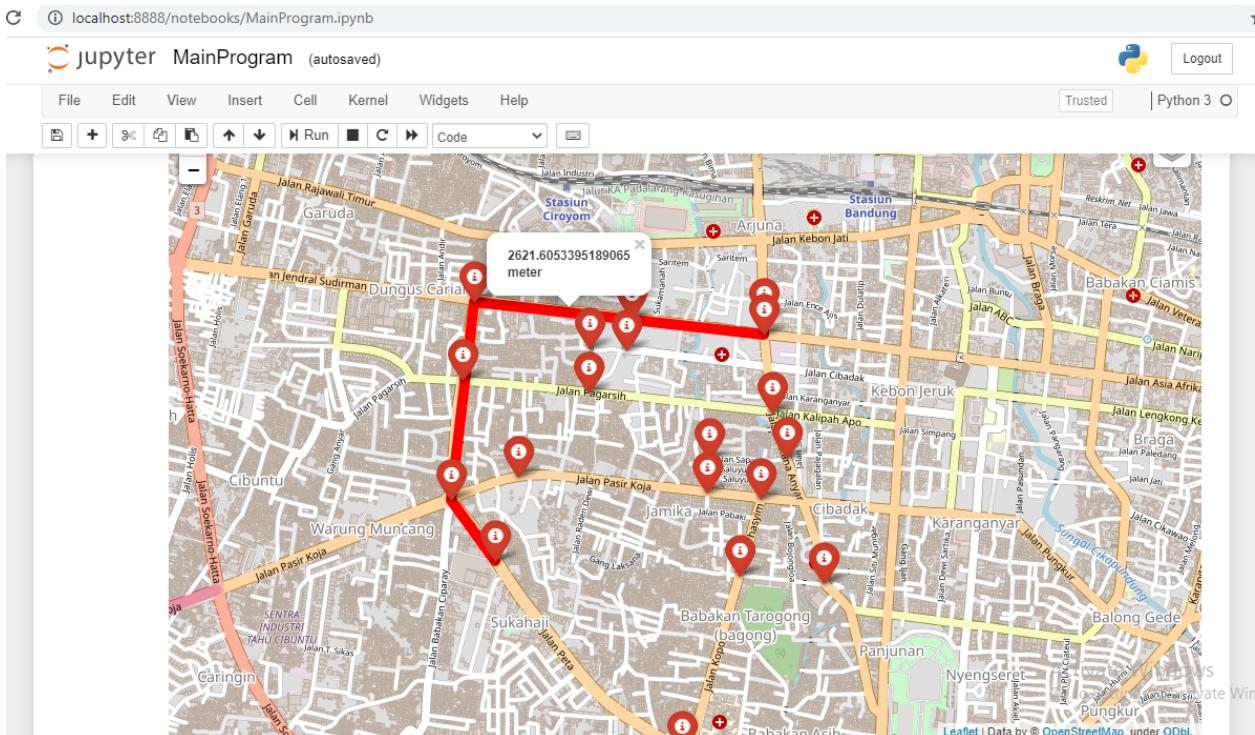
In [10]:

```
# Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda peta ^_^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal))):
    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
    simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal))):
    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
    simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^
Masukkan simpul awal: Gerbang_depan
Tidak terdapat simpul Gerbang_depan, silahkan masukkan kembali input simpul awal
Masukkan simpul awal: SMANOP
Masukkan simpul tujuan: ppp
Tidak terdapat simpul ppp, silahkan masukkan kembali input simpul tujuan
Masukkan simpul tujuan: FCL
Masukkan simpul tujuan: FCL
Shortest path dari SMANOP ke FCL adalah
SMANOP -> JA -> JC -> JJ -> Perliman -> FCL
Dengan total jarak adalah 2621.6053395189065 meter
```

In [7]:

```
# Cell 4, untuk memvisualisasikan jalur terpendek pada map
# Klik pada jalur untuk melihat jaraknya
coorSol = []
```



Berikut adalah hasil pencarian dari AP ke JJ

jupyter MainProgram (unsaved changes)

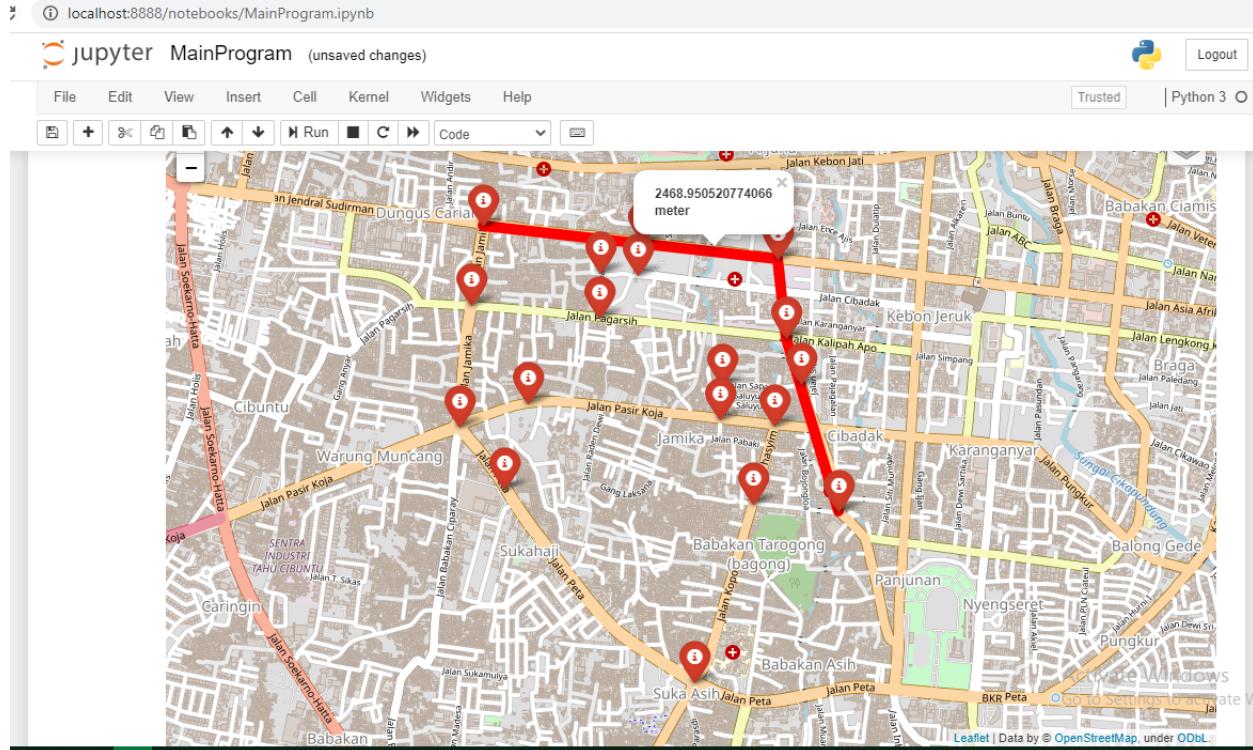
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```

In [12]: # Cell 3, untuk menerima input simpul awal dan simpul goal untuk dicari shortest pathnya dengan algoritma A*
# CATATAN: Nama simpul dapat diketahui dengan melakukan klik pada penanda peta
# Setelah nama simpul awal dan goal diinputkan, maka akan ditampilkan jalurnya dengan menggunakan text
# Visualisasi jalur terdapat pada cell 4
print("Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^")
simpul_awal = str(input("Masukkan simpul awal: "))
while(not(graf_peta.isVertex(simpul_awal))):
    print("Tidak terdapat simpul " + simpul_awal + ", silahkan masukkan kembali input simpul awal")
    simpul_awal = str(input("Masukkan simpul awal: "))
simpul_goal = str(input("Masukkan simpul tujuan: "))
while(not(graf_peta.isVertex(simpul_goal))):
    print("Tidak terdapat simpul " + simpul_goal + ", silahkan masukkan kembali input simpul tujuan")
    simpul_goal = str(input("Masukkan simpul tujuan: "))
graf_peta.findShortestPath(simpul_awal, simpul_goal)

Untuk mengetahui nama simpul silahkan klik pada penanda di Peta ^_^
Masukkan simpul awal: AP
Masukkan simpul tujuan: JJ
Shortest path dari AP ke JJ adalah
AP -> PS -> PA -> JA -> JC -> JJ
Dengan total jarak adalah 2468.950520774066 meter

```



BAB V

KESIMPULAN, SARAN, REFLEKSI DAN KOMENTAR

I. Kesimpulan

Algoritma A* (A star) merupakan algoritma yang optimal untuk mencari lintasan terpendek pada peta. Hal ini disebabkan oleh heuristic evaluation function-nya. Heuristic evaluation function $f(n)$ pada algoritma A* merupakan estimasi total biaya dari suatu lintasan yang melalui n ke tujuan. $f(n) = g(n) + h(n)$. $g(n)$ merupakan (estimasi) total biaya dari titik asal ke titik n dan $h(n)$ merupakan estimasi total biaya dari titik n ke titik tujuan. Pada permasalahan ini, heuristic function yang digunakan adalah straight line distance sehingga estimasi biaya $h(n)$ selalu lebih kecil dari biaya $h^*(n)$ yang sesungguhnya.

II. Saran

Untuk pengembangan lebih lanjut, bisa digunakan Google Maps API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta.

III. Refleksi dan Komentar

Pada tugas kecil 3 ini kami tau bagaimana mencari lintasan dari peta, dari mulai membuat graf pada peta, kemudian mencari lintasan terpendek antara simpul awal dan simpul goal.

DAFTAR PUSTAKA

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

<https://medium.com/@priagungkhusuma/cara-memvisualisasikan-peta-menggunakan-folium-python-b7d4a4c50cb3>

<https://analyticsvidhya.com/blog/2020/06/guide-geospatial-analysis-folium-python>

<https://informa.poltekindonusa.ac.id/index.php/informa/article/view/74/68>