Laporan Tugas Kecil III IF2211 Strategi Algoritma
Semester II Tahun 2020/2021
Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Disusun Oleh :

Jesica (13519011)
Rhea Elka Pandumpi (13519047)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2020

1. Kode Program

Kelas Graf

```
class Graf:
    def __init__(self, Nsimp):

        # menyimpan jumlah simpul
        self.Nsimp = Nsimp

        # menyimpan nama simpul sbg dictionary
        self.dict = {}

        # graf disimpan dalam bentuk dictionary
        self.graph = {}

        # menyimpan data koordinat setiap simpul
        self.koor = {}

        # menambahkan simpul dalam graf
        # inisialisasi array sebanyak Nsimp
        for i in range(Nsimp):
            self.graph[i] = []

    def getNSimpul(self):
        return self.Nsimp

    def getDict(self):
        return self.dict

    def getGraf(self):
        return self.graph

    def getKoor(self):
        return self.koor

    def add_edge(self, v1, v2, bobot):
        # diasumsikan simpul sudah terdapat dalam graf
        temp = [v2, bobot]
        if(temp not in self.graph[v1]):
            self.graph[v1].append(temp)
```

```
        temp1 = [v1, bobot]
        if(temp1 not in self.graph[v2]):
            self.graph[v2].append(temp1)

    def haversineDistance(self, index1, index2):
        EARTH_RAD=6371
        # latitudes and longitudes
        # lokasi dari 2 titik di koordinat bola (lintang dan bujur)
        lat1 = self.koor[index1][0]
        long1 = self.koor[index1][1]
        lat2 = self.koor[index2][0]
        long2 = self.koor[index2][1]
        # distance between latitudes and longitudes
        dLat = radians(lat2-lat1)
        dLong = radians(long2-long1)
        # formula
        haver = sin(dLat/2)**2+cos(radians(lat1))*cos(radians(lat2))*sin(dLong/2)**2
        haver = 2* atan2 (sqrt(haver), sqrt(1-haver))
        haver = EARTH_RAD*haver
        return haver

    def generateGraphfromMap(self, edges, mapkoor):
        self.koor = mapkoor
        self.dict = None

        for i in range(len(edges)):
            index1 = edges[i][0]
            index2 = edges[i][1]
            e = self.haversineDistance(index1,index2)
            self.add_edge(index1,index2,e)
```

Pembacaan Graf

```
# untuk membaca input graf yang berasal dari map
def readData(edges, coors):
    n = len(coors)
    key = [0 for i in range(n)]
    for i in range(n):
        key[i] = i
```

```
    mapkoor = {}
    for nomor, xy in zip(key,coors):
        mapkoor[nomor] = xy
    #print(mapkoor)

    # buat grafnya
    g = Graf(n)
    g.generateGraphfromMap(edges,mapkoor)
    return g
```

Algoritma A-Star untuk Route Planning

```
def findBest(arr, open):
    best = open[0]
    for i in (open):
        if arr[i]<arr[best]:
            best = i
    return best

def findNeighbors(graph,curr):
    neighbors = []
    for i in (graph.getGraf()[curr]):
        neighbors.append(i[0])
    return neighbors

def displayPath(cameFrom,curr,cost):
    result = [curr]
    while cameFrom[curr] != -999:
        curr = cameFrom[curr]
        result.insert(0,curr)
    result.append(cost)
    return result

def AStar(g,startNode,goalNode):
    toEvaluate = [startNode] # to be evaluated
    evaluated = [] # already evaluated
    # Total cost
    fcost = {}
    for i in range(len(g.getGraf())):
        fcost[i] = g.haversineDistance(i,goalNode)
    # Initialize cost
```

```
    gcost = [999] * len(g.getGraf()) # from start to current Node
    gcost[startNode] = 0
    # initialize array
    cameFrom = [-999] * len(g.getGraf())
    while(toEvaluate): # Not yet empty
        # Find the minimum
        currentNode = findBest(fcost,toEvaluate)
        if(currentNode==goalNode):
            # Found
            return(displayPath(cameFrom,currentNode,gcost[currentNode]))
        else:
            # remove from the list to be evaluated
            toEvaluate.remove(currentNode)
            evaluated.insert(0,currentNode)
            neighbors = findNeighbors(g,currentNode)
            for neighbor in neighbors:
                if not (neighbor in evaluated): # not yet evaluated
                    if not (neighbor in toEvaluate): # no duplicates
                        toEvaluate.append(neighbor)
                    temporaryCost = gcost[currentNode] +
g.haversineDistance(currentNode,neighbor)
                    # finding a better path
                    if temporaryCost < gcost[neighbor]:
                        cameFrom[neighbor] = currentNode
                        gcost[neighbor] = temporaryCost
                        fcost[neighbor]=gcost[neighbor]+g.haversineDistance(neighbor,goalNode)
    return None
```

Memperoleh *key* yang dimiliki oleh suatu *value*

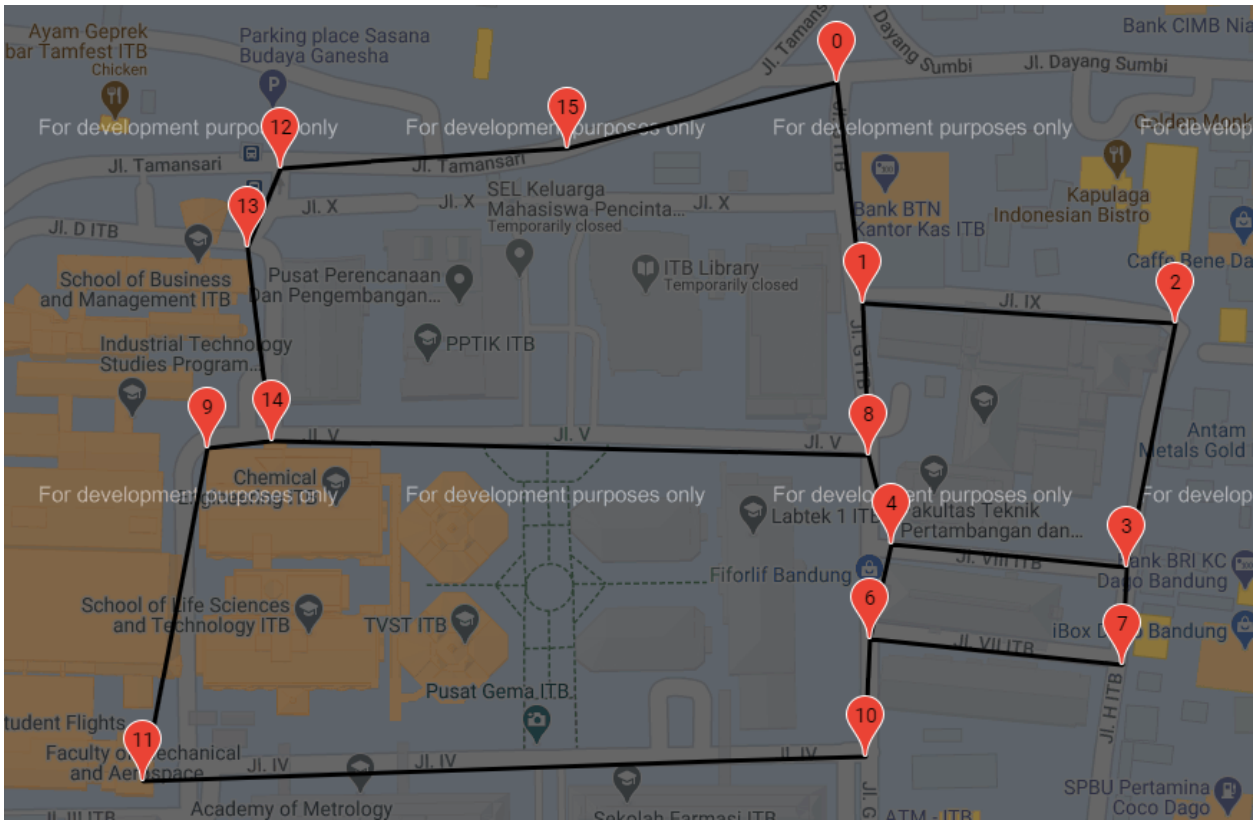
```
def getKey(my_dict, val):
    # list out keys and values separately
    key_list = list(my_dict.keys())
    val_list = list(my_dict.values())

    print(key_list,val_list)

    # print key with val 100
    position = val_list.index(val)
    return(key_list[position])
```

Kode main2.py

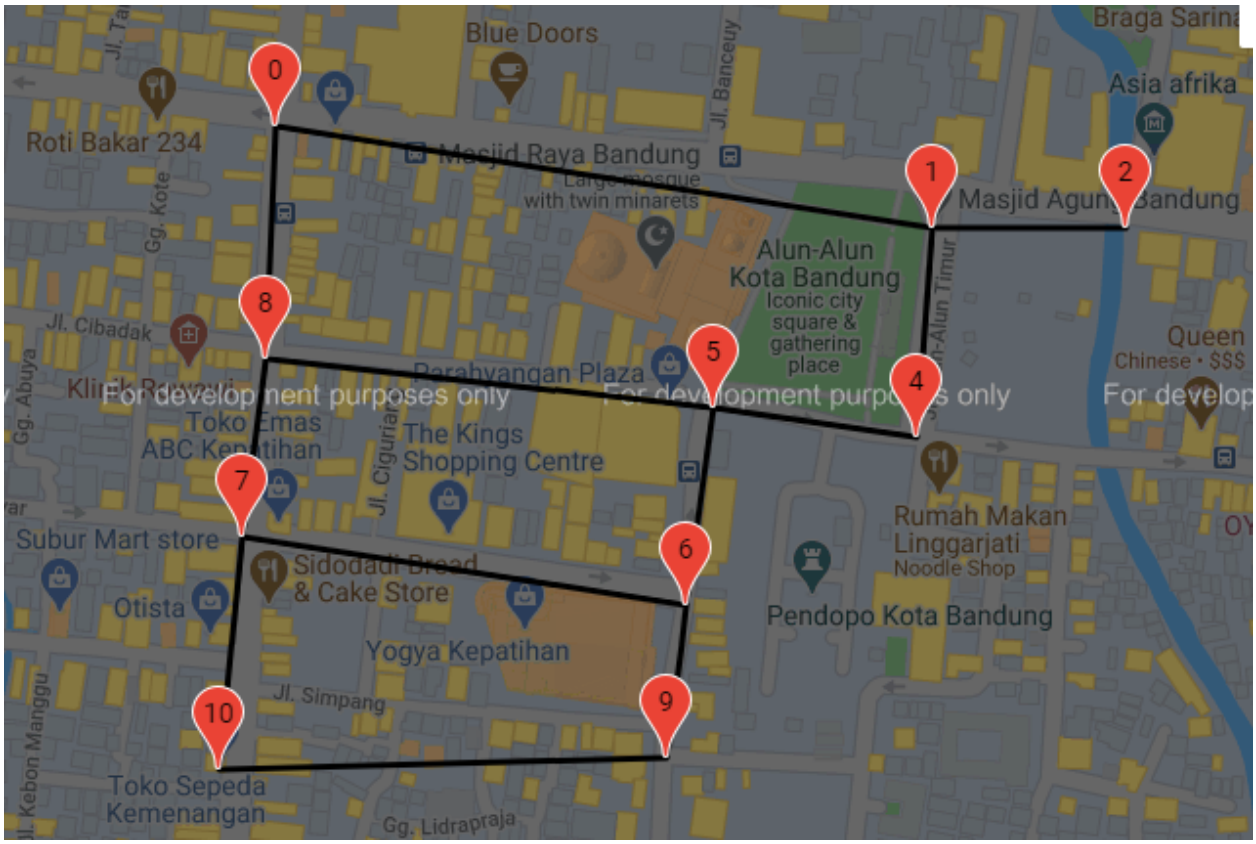
```
from flask import *
import sys
import json
from read import *
app = Flask(__name__)

@app.route("/")
def launch():
    return render_template("map.html")

@app.route('/postmethod', methods = ['POST'])
def get_post_javascript_data():
    jsdata = request.get_json()
    arrOfCoords = []
    arrOfEdges = []
    for i in range(len(jsdata['coordinates'])):
        lat = jsdata['coordinates'][i]['lat']
        lng = jsdata['coordinates'][i]['lng']
        arrOfCoords.append([lat,lng])
    arrOfEdges = jsdata['edges']
    g = readData(arrOfEdges,arrOfCoords)
    start = int(jsdata['start'])
    goal = int(jsdata['finish'])
    sol = AStar(g,start,goal)
    cost = sol.pop()
    solution = {'solution': sol, 'cost': str("{:.4f}".format(cost)) + " km"}
    return json.dumps(solution)

if __name__ == "__main__":
    app.run()
```

2. **Peta graf/input**

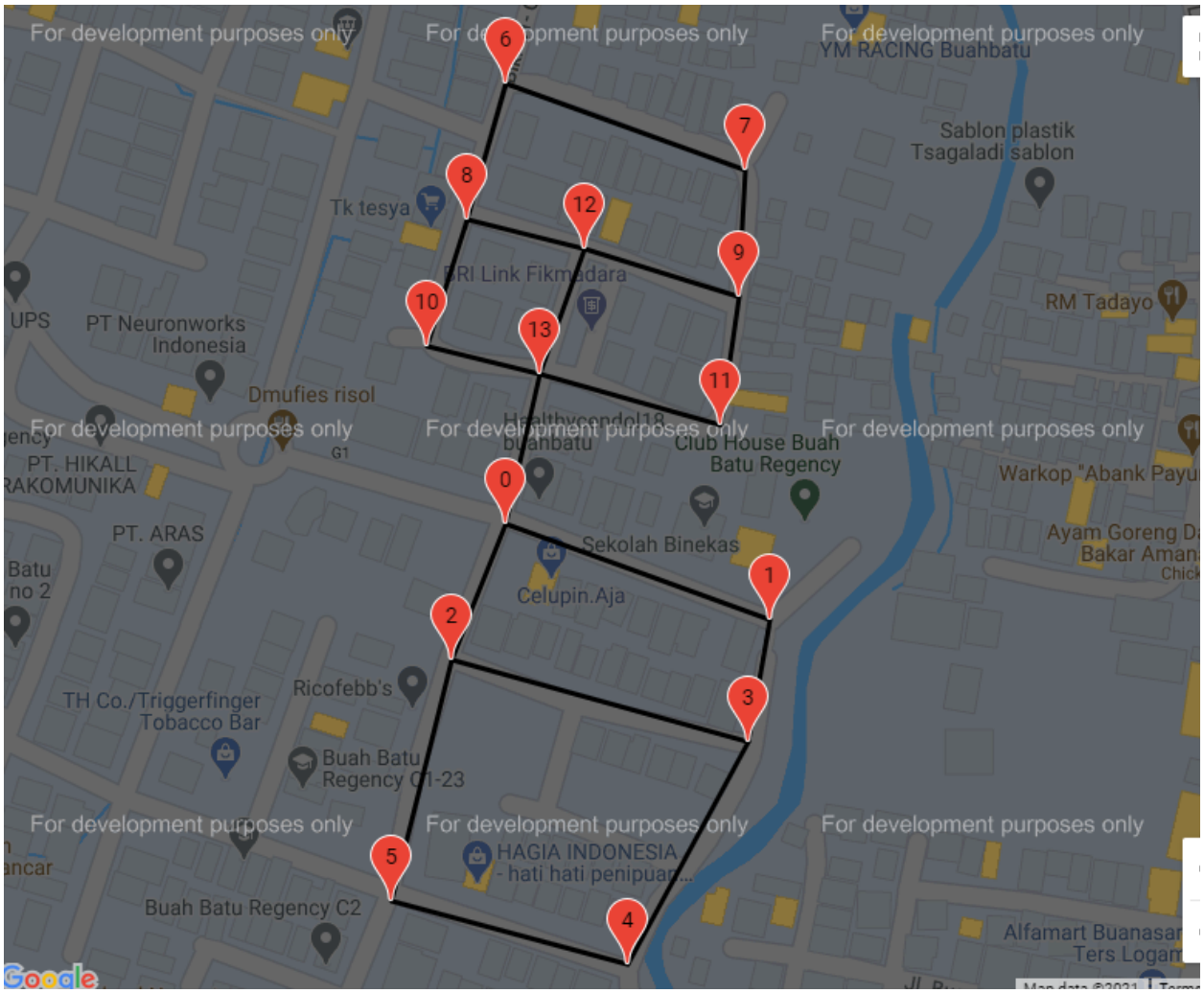
Daerah kampus ITB



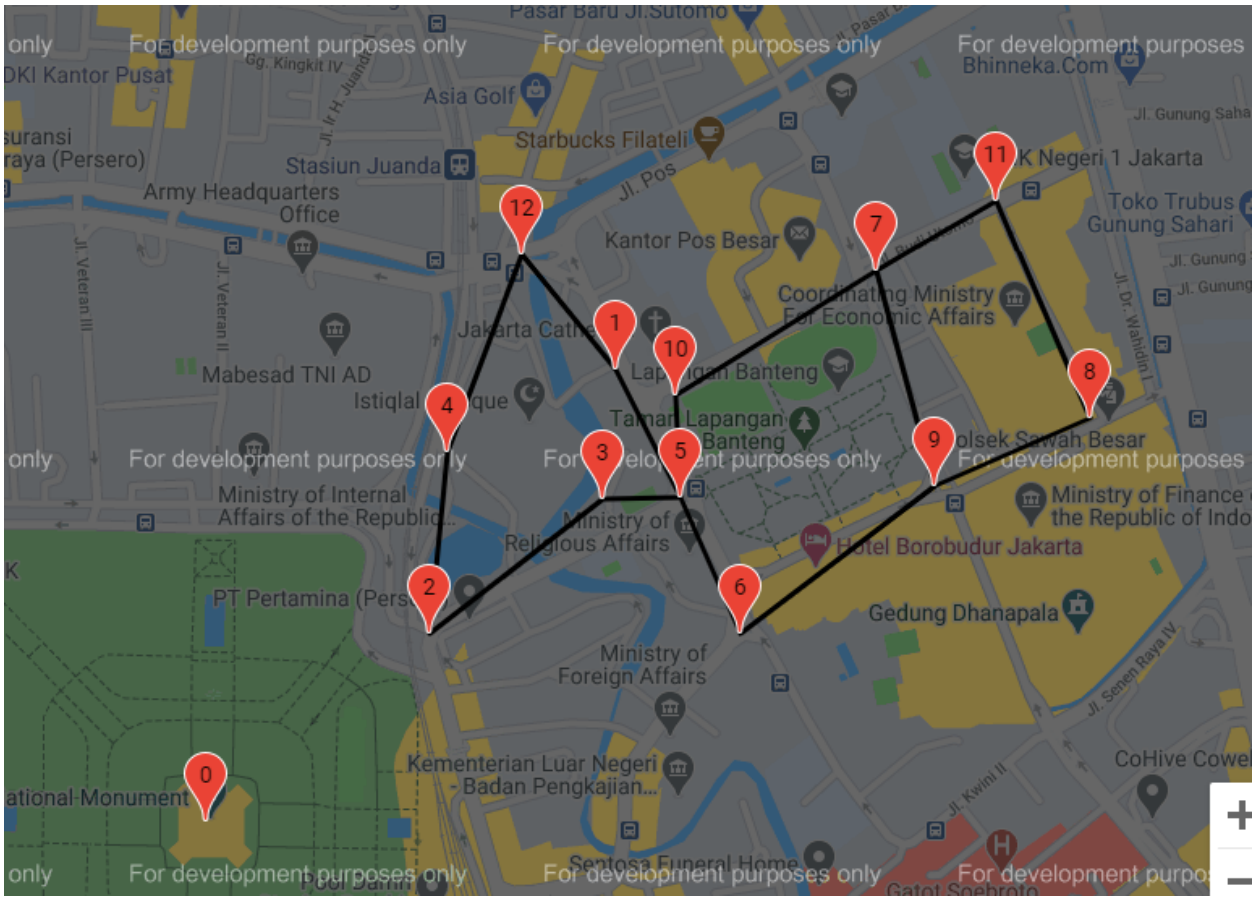
Daerah Alun-alun Bandung



Daerah Buahbatu

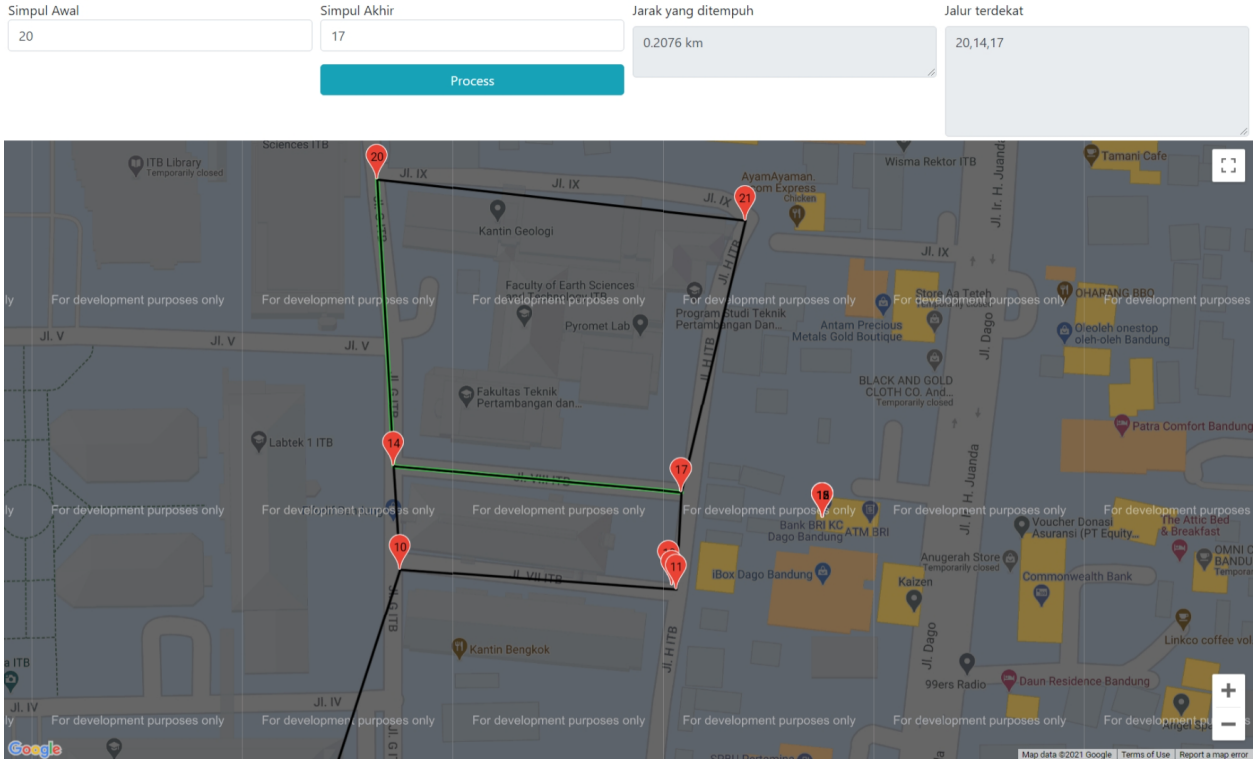


Daerah sekitar Monas



3. **Screenshot peta yang memperlihatkan lintasan terpendek untuk sepasang simpul**

Dari simpul 20 ke 17



Dari simpul 1 ke simpul 6

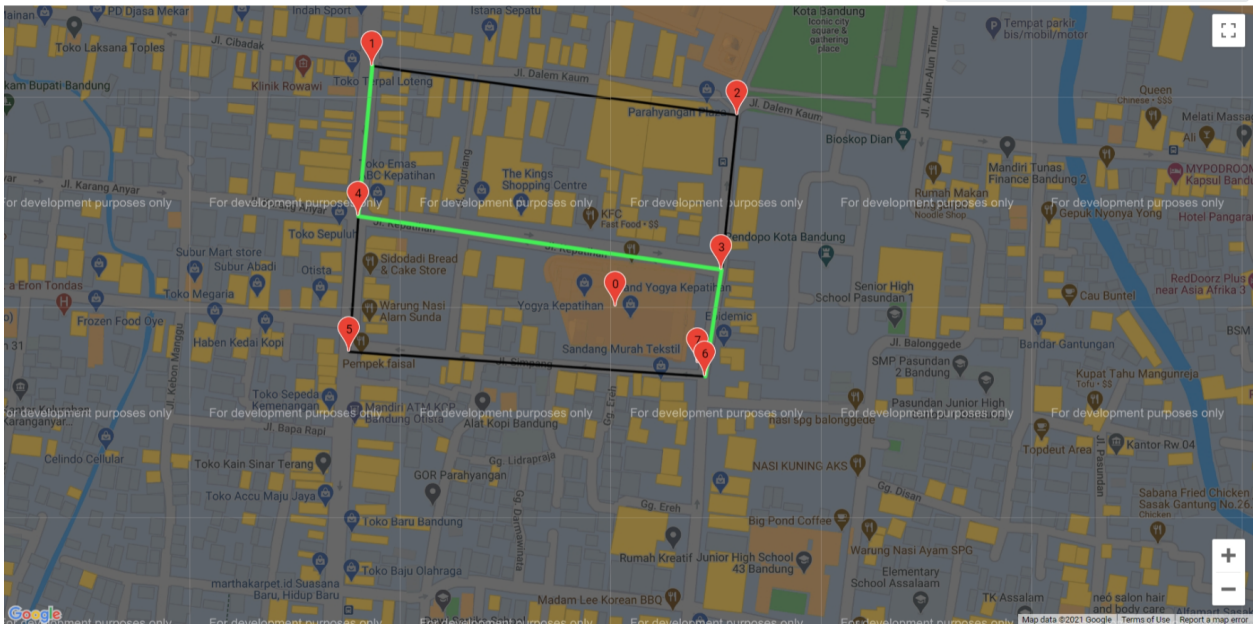
Simpul Awal: 1

Simpul Akhir: 6

Process

Jarak yang ditempuh: 0.4513 km

Jalur terdekat: 1,4,3,6



Dari simpul 8 ke simpul 0

Simpul Awal: 8

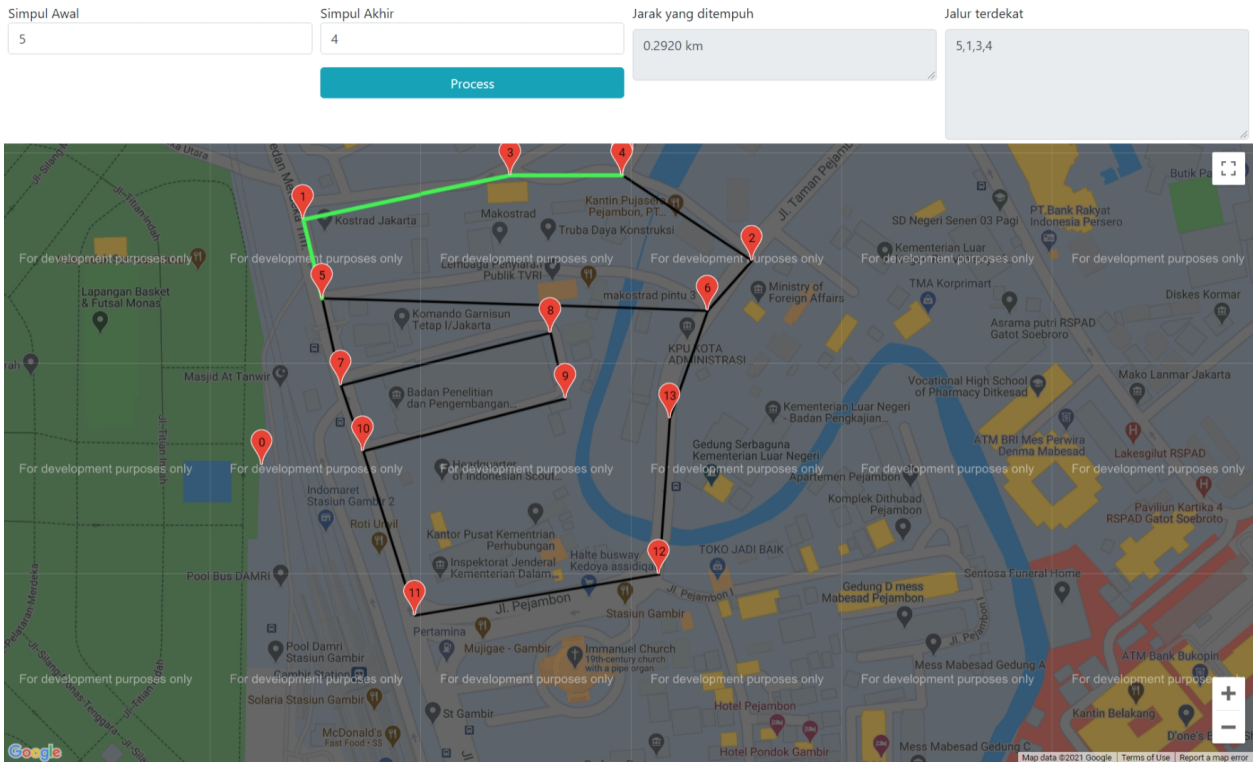
Simpul Akhir: 0

Process

Jarak yang ditempuh: 0.1354 km

Jalur terdekat: 8,7,3,0

Dari simpul 5 ke simpul 4



4. **Alamat tempat kode dikumpulkan**
   https://tiny.cc/KumpulTucilStima3

Tabel

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