LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Daffa Ananda Pratama Resyaly

13519107

Kelas 02

Sekolah Teknik Elektro dan Informatika

Program Studi Teknik Informatika

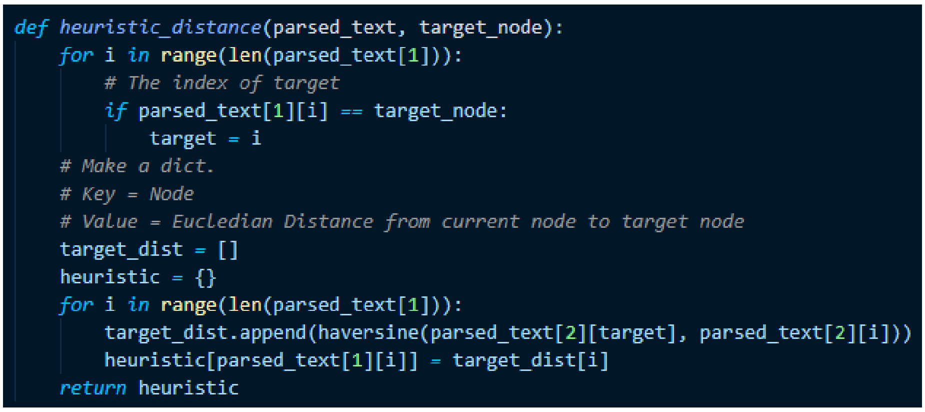
Tahun Ajaran 2020/2021

I. Kode Program

Pada *source code* dari program yang telah dibuat, terdapat empat buah file python dan terdapat total sebuah program utama dan sepuluh buah fungsi, termasuk enam buah fungsi utama dan empat buah fungsi tambahan.

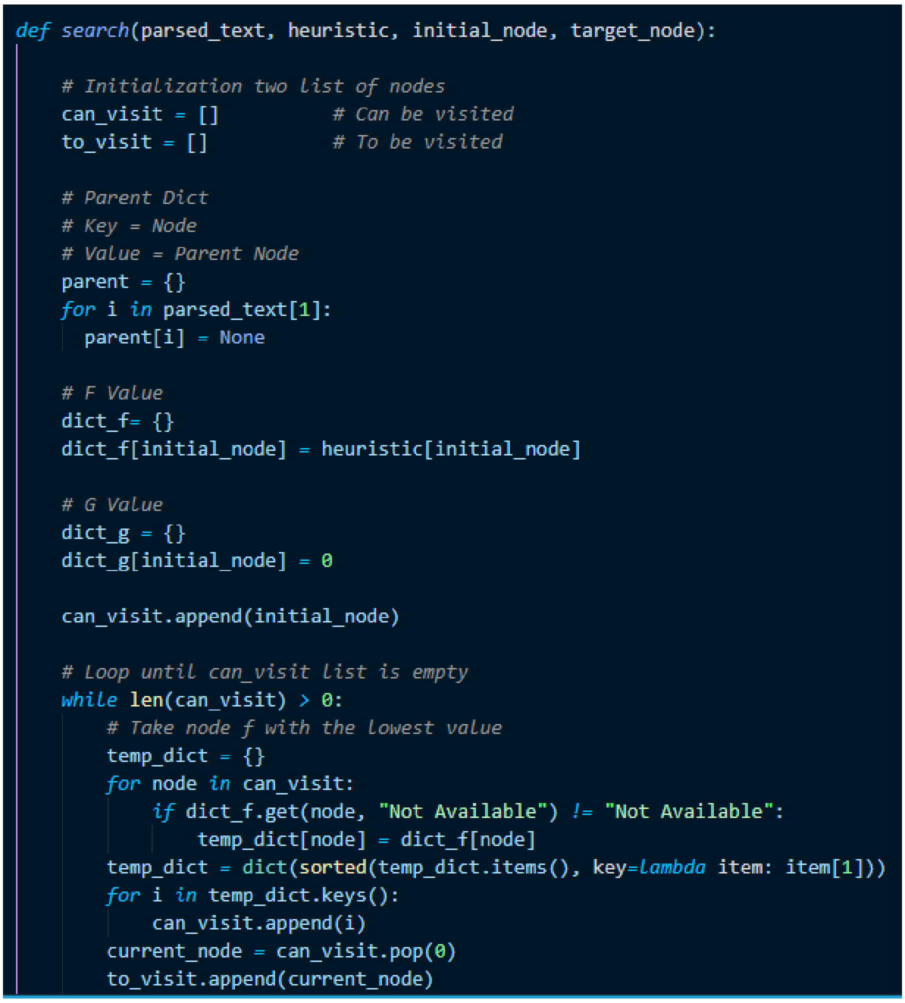
1.1. Fungsi Utama

1.1.1. Fungsi untuk menghitung *heuristic distance* dari teks file

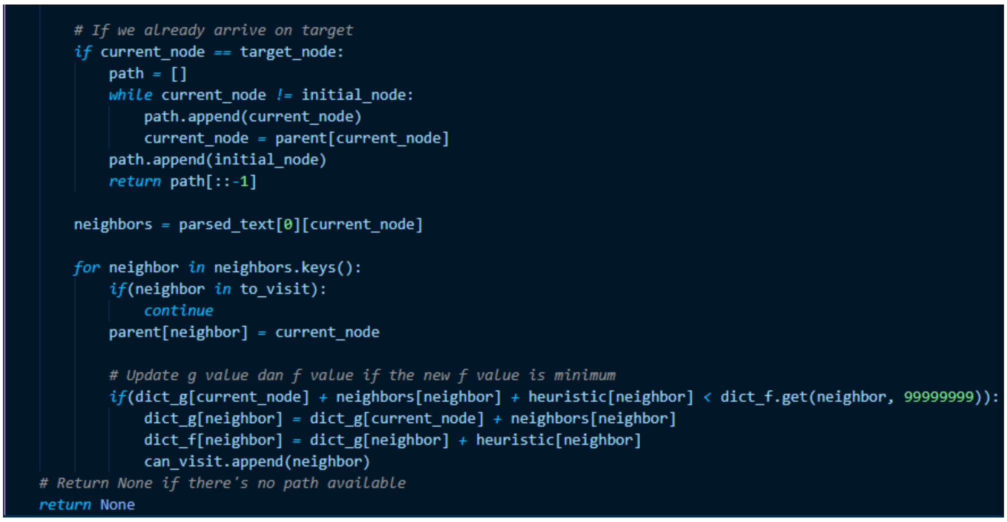


```python
def heuristic_distance(parsed_text, target_node):
    for i in range(len(parsed_text[1])):
        # The index of target
        if parsed_text[1][i] == target_node:
            target = i
    # Make a dict.
    # Key = Node
    # Value = Eucledian Distance from current node to target node
    target_dist = []
    heuristic = {}
    for i in range(len(parsed_text[1])):
        target_dist.append(haversine(parsed_text[2][target], parsed_text[2][i]))
        heuristic[parsed_text[1][i]] = target_dist[i]
    return heuristic
```

1.1.2. Fungsi untuk mencari jalur terpendek dari simpul asal ke simpul tujuan

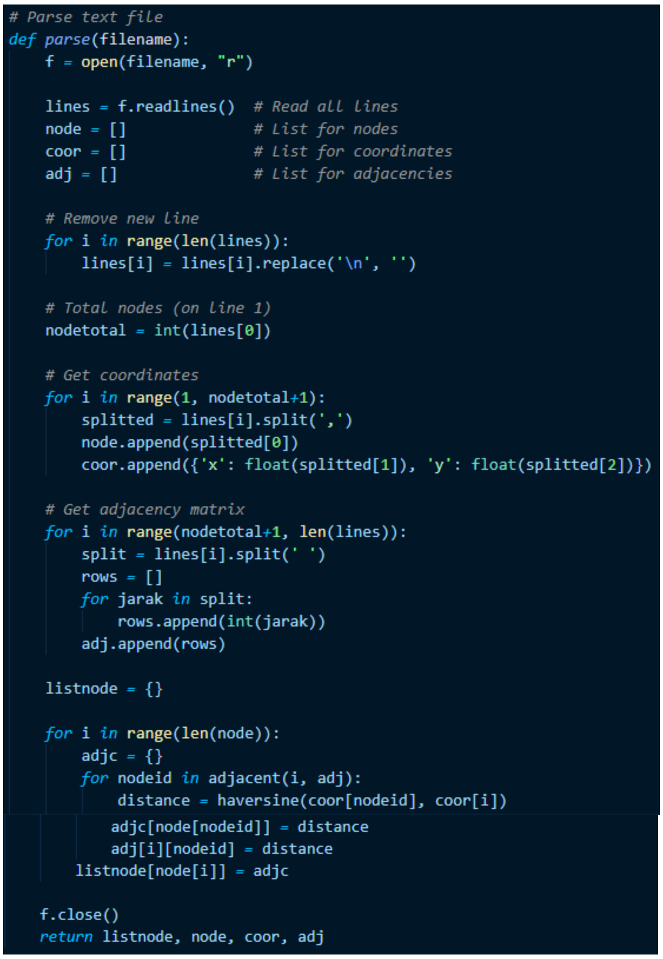


```python
def search(parsed_text, heuristic, initial_node, target_node):

    # Initialization two list of nodes
    can_visit = []          # Can be visited
    to_visit = []           # To be visited

    # Parent Dict
    # Key = Node
    # Value = Parent Node
    parent = {}
    for i in parsed_text[1]:
        parent[i] = None

    # F Value
    dict_f= {}
    dict_f[initial_node] = heuristic[initial_node]

    # G Value
    dict_g = {}
    dict_g[initial_node] = 0

    can_visit.append(initial_node)

    # Loop until can_visit list is empty
    while len(can_visit) > 0:
        # Take node f with the lowest value
        temp_dict = {}
        for node in can_visit:
            if dict_f.get(node, "Not Available") != "Not Available":
                temp_dict[node] = dict_f[node]
        temp_dict = dict(sorted(temp_dict.items(), key=lambda item: item[1]))
        for i in temp_dict.keys():
            can_visit.append(i)
        current_node = can_visit.pop(0)
        to_visit.append(current_node)
```



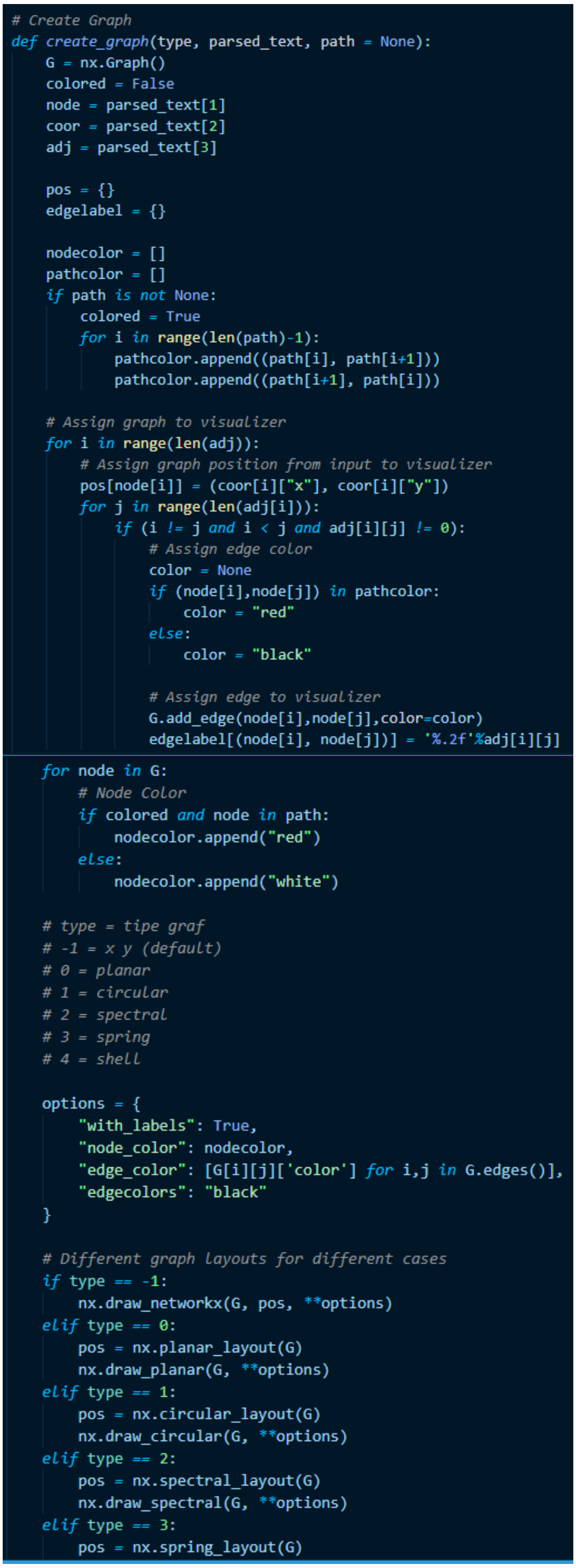
```python
        # If we already arrive on target
        if current_node == target_node:
            path = []
            while current_node != initial_node:
                path.append(current_node)
                current_node = parent[current_node]
            path.append(initial_node)
            return path[::-1]

        neighbors = parsed_text[0][current_node]

        for neighbor in neighbors.keys():
            if(neighbor in to_visit):
                continue
            parent[neighbor] = current_node

            # Update g value dan f value if the new f value is minimum
            if(dict_g[current_node] + neighbors[neighbor] + heuristic[neighbor] < dict_f.get(neighbor, 99999999)):
                dict_g[neighbor] = dict_g[current_node] + neighbors[neighbor]
                dict_f[neighbor] = dict_g[neighbor] + heuristic[neighbor]
                can_visit.append(neighbor)
    # Return None if there's no path available
    return None
```

1.1.3. Fungsi untuk membaca file teks dan melakukan *parsing* teks dari file



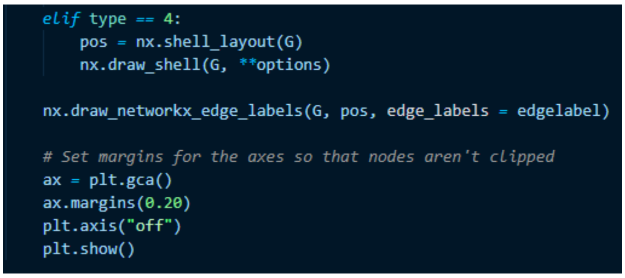
```python
# Parse text file
def parse(filename):
    f = open(filename, "r")

    lines = f.readlines()  # Read all lines
    node = []              # List for nodes
    coor = []              # List for coordinates
    adj = []               # List for adjacencies

    # Remove new line
    for i in range(len(lines)):
        lines[i] = lines[i].replace('\n', '')

    # Total nodes (on line 1)
    nodetotal = int(lines[0])

    # Get coordinates
    for i in range(1, nodetotal+1):
        splitted = lines[i].split(',')
        node.append(splitted[0])
        coor.append({'x': float(splitted[1]), 'y': float(splitted[2])})

    # Get adjacency matrix
    for i in range(nodetotal+1, len(lines)):
        split = lines[i].split(' ')
        rows = []
        for jarak in split:
            rows.append(int(jarak))
        adj.append(rows)

    listnode = {}

    for i in range(len(node)):
        adjc = {}
        for nodeid in adjacent(i, adj):
            distance = haversine(coor[nodeid], coor[i])
            adjc[node[nodeid]] = distance
            adj[i][nodeid] = distance
        listnode[node[i]] = adjc

    f.close()
    return listnode, node, coor, adj
```

1.1.4. Fungsi untuk membuat graf dari simpul dan koordinat yang ada



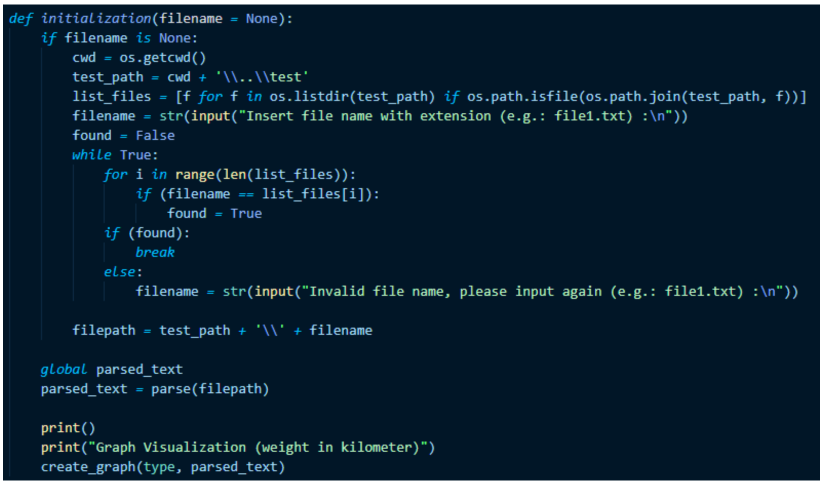
```python
# Create Graph
def create_graph(type, parsed_text, path = None):
    G = nx.Graph()
    colored = False
    node = parsed_text[1]
    coor = parsed_text[2]
    adj = parsed_text[3]

    pos = {}
    edgelabel = {}

    nodecolor = []
    pathcolor = []
    if path is not None:
        colored = True
        for i in range(len(path)-1):
            pathcolor.append((path[i], path[i+1]))
            pathcolor.append((path[i+1], path[i]))

    # Assign graph to visualizer
    for i in range(len(adj)):
        # Assign graph position from input to visualizer
        pos[node[i]] = (coor[i]["x"], coor[i]["y"])
        for j in range(len(adj[i])):
            if (i != j and i < j and adj[i][j] != 0):
                # Assign edge color
                color = None
                if (node[i],node[j]) in pathcolor:
                    color = "red"
                else:
                    color = "black"

                # Assign edge to visualizer
                G.add_edge(node[i],node[j],color=color)
                edgelabel[(node[i], node[j])] = '%.2f'%adj[i][j]
```

```python
    for node in G:
        # Node Color
        if colored and node in path:
            nodecolor.append("red")
        else:
            nodecolor.append("white")

    # type = tipe graf
    # -1 = x y (default)
    # 0 = planar
    # 1 = circular
    # 2 = spectral
    # 3 = spring
    # 4 = shell

    options = {
        "with_labels": True,
        "node_color": nodecolor,
        "edge_color": [G[i][j]['color'] for i,j in G.edges()],
        "edgecolors": "black"
    }

    # Different graph layouts for different cases
    if type == -1:
        nx.draw_networkx(G, pos, **options)
    elif type == 0:
        pos = nx.planar_layout(G)
        nx.draw_planar(G, **options)
    elif type == 1:
        pos = nx.circular_layout(G)
        nx.draw_circular(G, **options)
    elif type == 2:
        pos = nx.spectral_layout(G)
        nx.draw_spectral(G, **options)
    elif type == 3:
        pos = nx.spring_layout(G)
```



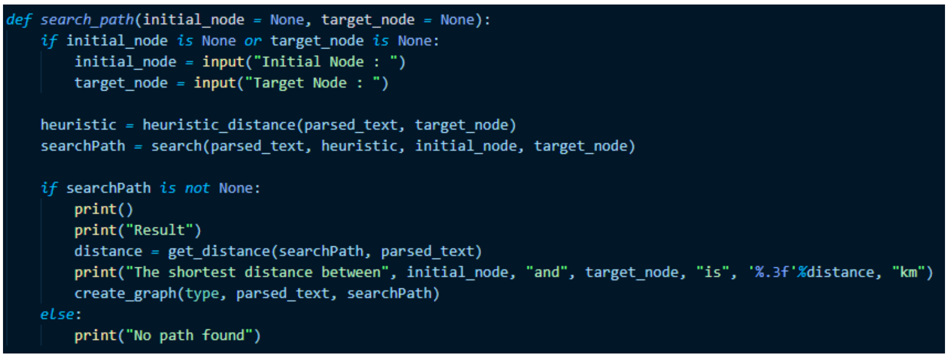
```
    elif type == 4:
        pos = nx.shell_layout(G)
        nx.draw_shell(G, **options)

    nx.draw_networkx_edge_labels(G, pos, edge_labels = edgelabel)

    # Set margins for the axes so that nodes aren't clipped
    ax = plt.gca()
    ax.margins(0.20)
    plt.axis("off")
    plt.show()
```

1.1.5. Fungsi untuk menginisialisasi program dan menampilkan visualisasi graf



```
def initialization(filename = None):
    if filename is None:
        cwd = os.getcwd()
        test_path = cwd + '\\..\\test'
        list_files = [f for f in os.listdir(test_path) if os.path.isfile(os.path.join(test_path, f))]
        filename = str(input("Insert file name with extension (e.g.: file1.txt) :\n"))
        found = False
        while True:
            for i in range(len(list_files)):
                if (filename == list_files[i]):
                    found = True
            if (found):
                break
            else:
                filename = str(input("Invalid file name, please input again (e.g.: file1.txt) :\n"))

        filepath = test_path + '\\' + filename

    global parsed_text
    parsed_text = parse(filepath)

    print()
    print("Graph Visualization (weight in kilometer)")
    create_graph(type, parsed_text)
```

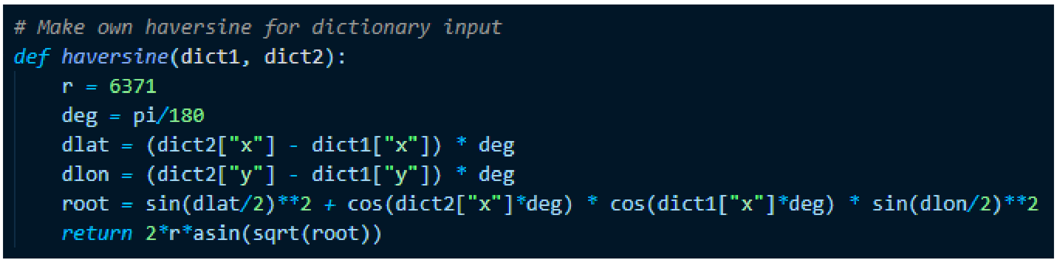
1.1.6. Fungsi untuk menampilkan hasil dari pencarian jalur terpendek antara kedua buah simpul



```
def search_path(initial_node = None, target_node = None):
    if initial_node is None or target_node is None:
        initial_node = input("Initial Node : ")
        target_node = input("Target Node : ")

    heuristic = heuristic_distance(parsed_text, target_node)
    searchPath = search(parsed_text, heuristic, initial_node, target_node)

    if searchPath is not None:
        print()
        print("Result")
        distance = get_distance(searchPath, parsed_text)
        print("The shortest distance between", initial_node, "and", target_node, "is", '%.3f'%distance, "km")
        create_graph(type, parsed_text, searchPath)
    else:
        print("No path found")
```

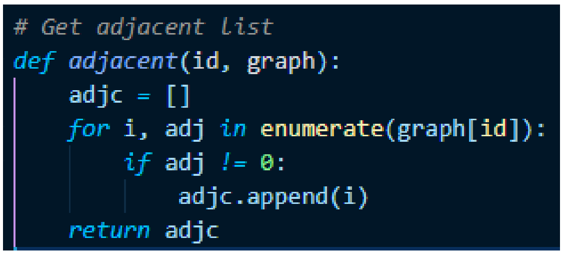
1.2. Fungsi Tambahan

1.2.1. Fungsi untuk menghitung hasil haversine dari dua buah *dictionary*



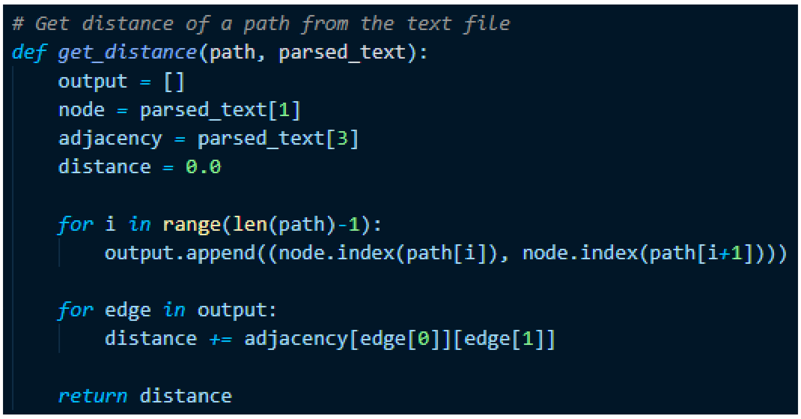
```python
# Make own haversine for dictionary input
def haversine(dict1, dict2):
    r = 6371
    deg = pi/180
    dlat = (dict2["x"] - dict1["x"]) * deg
    dlon = (dict2["y"] - dict1["y"]) * deg
    root = sin(dlat/2)**2 + cos(dict2["x"]*deg) * cos(dict1["x"]*deg) * sin(dlon/2)**2
    return 2*r*asin(sqrt(root))
```

1.2.2. Fungsi untuk mendapatkan list ketetanggaan tiap simpul



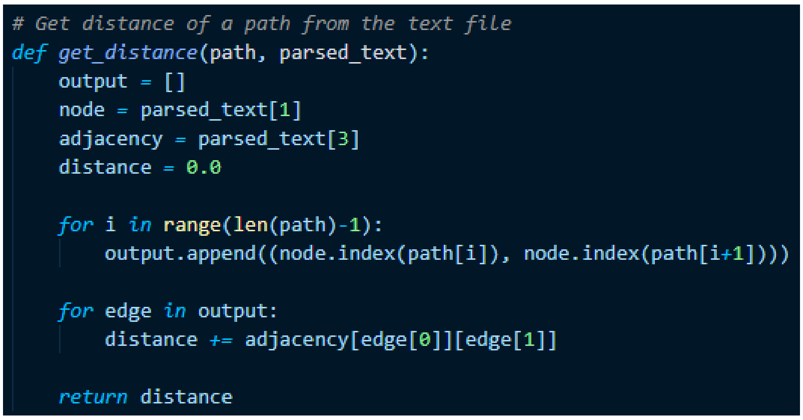
```python
# Get adjacent list
def adjacent(id, graph):
    adjc = []
    for i, adj in enumerate(graph[id]):
        if adj != 0:
            adjc.append(i)
    return adjc
```

1.2.3. Fungsi untuk mendapatkan jarak dari suatu jalur



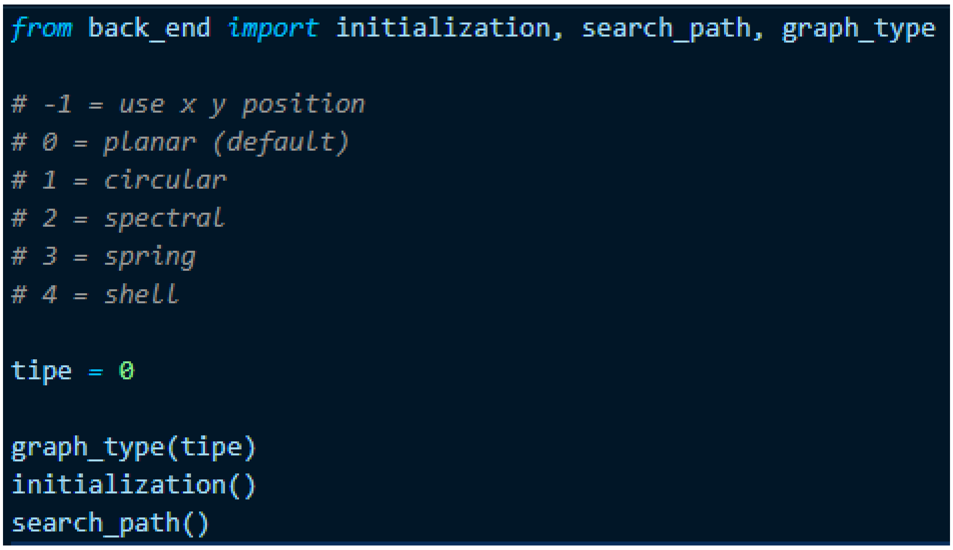
```python
# Get distance of a path from the text file
def get_distance(path, parsed_text):
    output = []
    node = parsed_text[1]
    adjacency = parsed_text[3]
    distance = 0.0

    for i in range(len(path)-1):
        output.append((node.index(path[i]), node.index(path[i+1])))

    for edge in output:
        distance += adjacency[edge[0]][edge[1]]

    return distance
```

1.2.4. Fungsi untuk menentukan jenis graf yang akan digunakan



```python
# Get distance of a path from the text file
def get_distance(path, parsed_text):
    output = []
    node = parsed_text[1]
    adjacency = parsed_text[3]
    distance = 0.0

    for i in range(len(path)-1):
        output.append((node.index(path[i]), node.index(path[i+1])))

    for edge in output:
        distance += adjacency[edge[0]][edge[1]]

    return distance
```

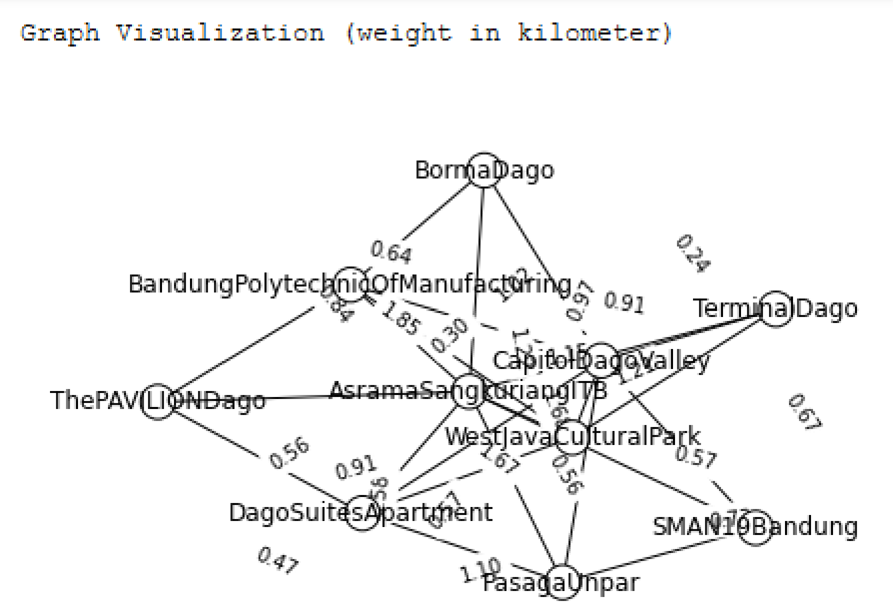
1.3. Program Utama



```python
from back_end import initialization, search_path, graph_type

# -1 = use x y position
# 0 = planar (default)
# 1 = circular
# 2 = spectral
# 3 = spring
# 4 = shell

tipe = 0

graph_type(tipe)
initialization()
search_path()
```

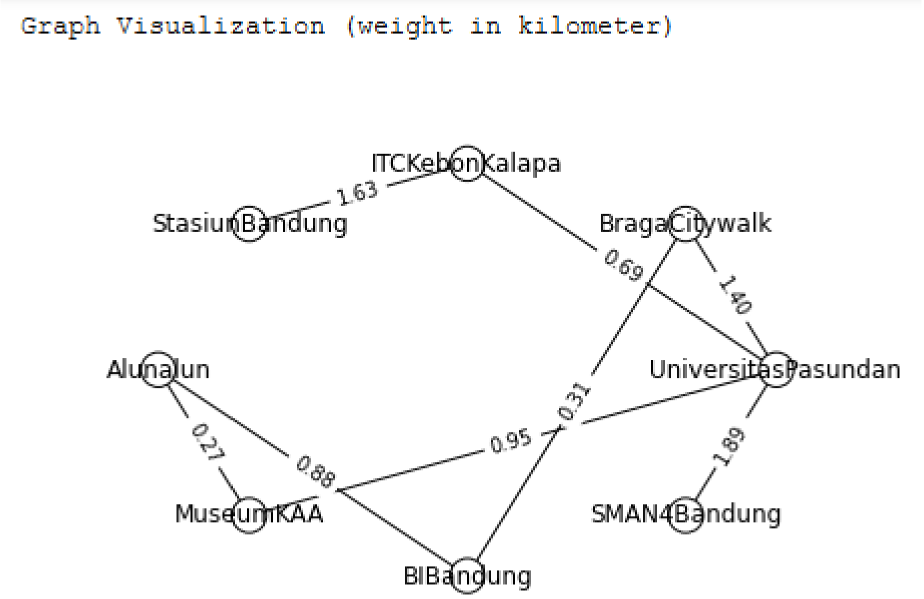
II. Peta / Graf Input

2.1. Graf Daerah Dago



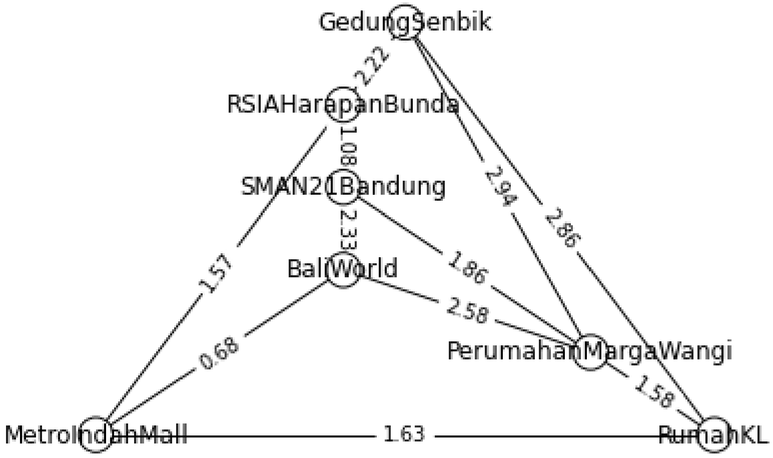
Graph Visualization (weight in kilometer)

2.2. Graf Daerah Alun-Alun Bandung

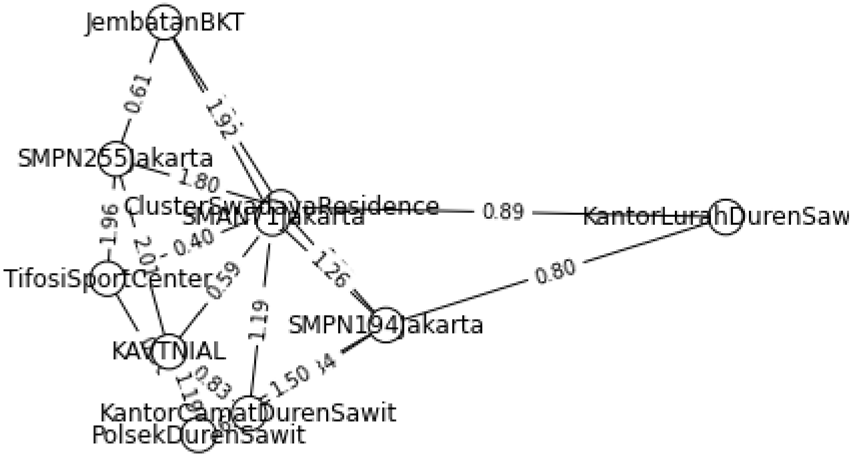


Graph Visualization (weight in kilometer)

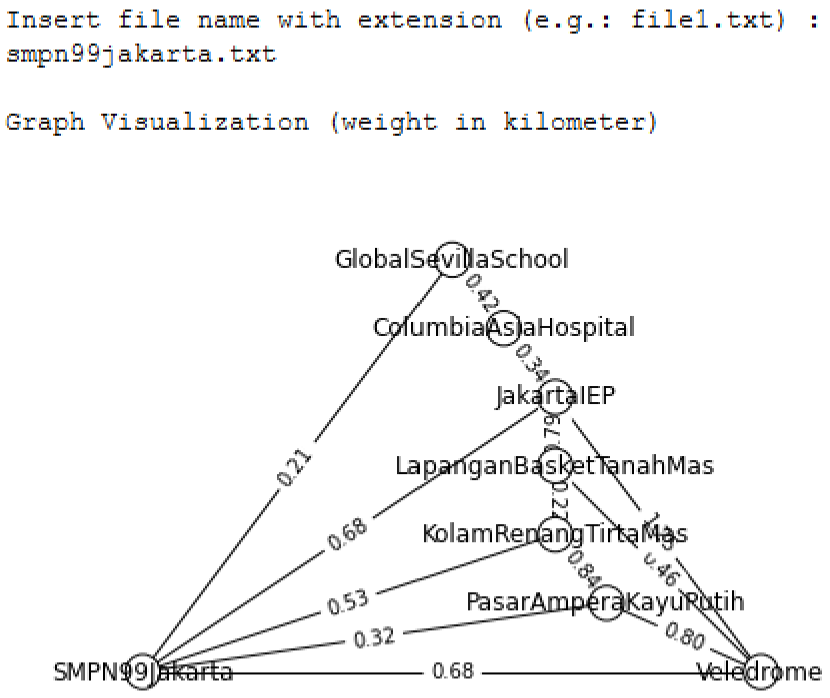
2.3. Graf Daerah BuahBatu



2.4. Graf Daerah Duren Sawit

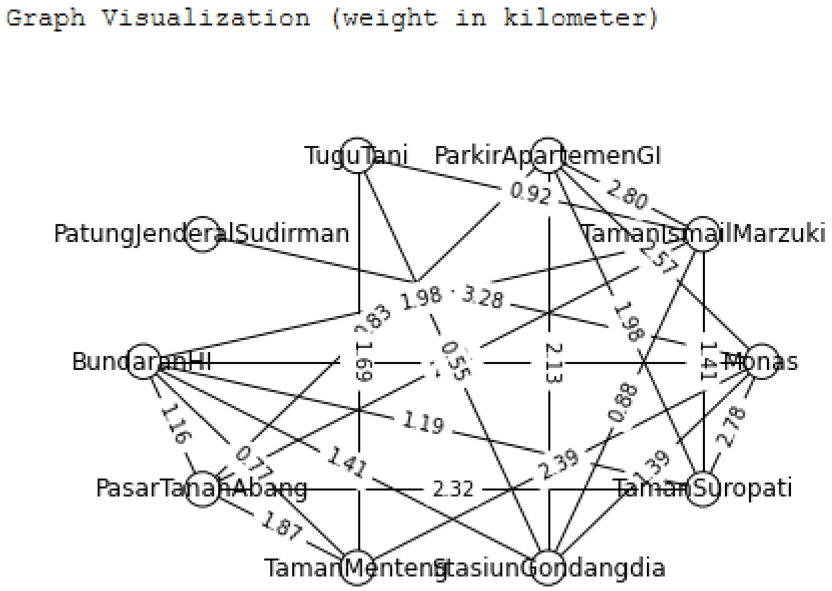


2.5. Graf Daerah SMPN 99 Jakarta



```
Insert file name with extension (e.g.: file1.txt) :
smpn99jakarta.txt

Graph Visualization (weight in kilometer)
```

2.6. Graf Daerah Bundaran HI

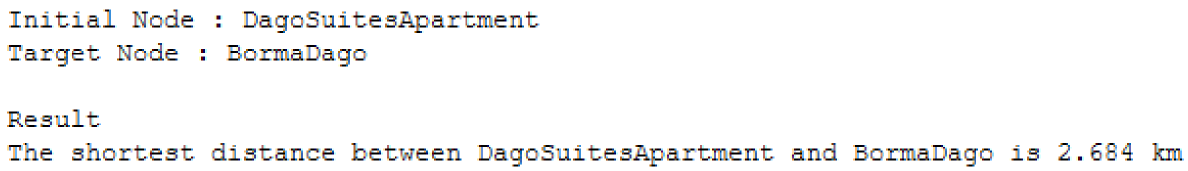


```
Graph Visualization (weight in kilometer)
```

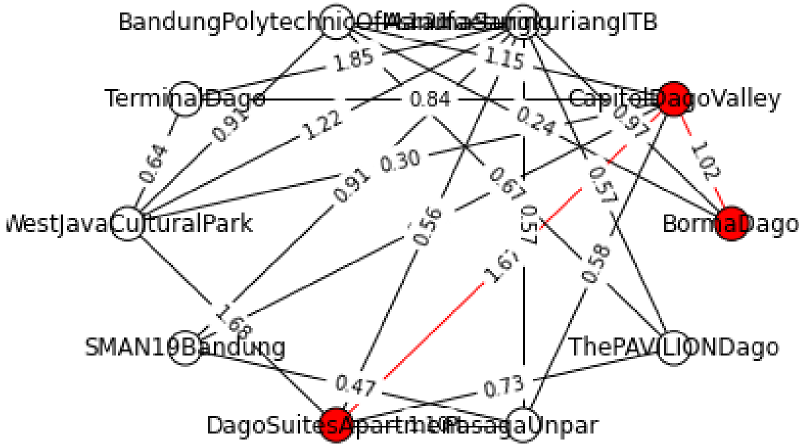
III. Screenshot Peta Lintasan Terpendek

3.1. Screenshot Lintasan Terpendek di Daerah Dago

Antara Dago Suites Apartment dan Borma Dago

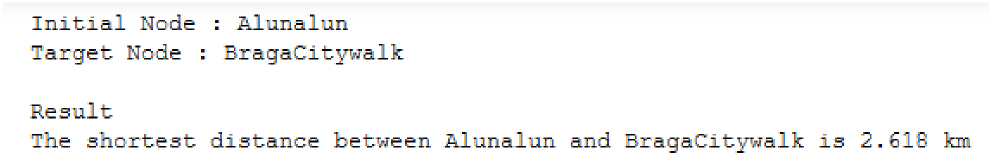


```
Initial Node : DagoSuitesApartment
Target Node : BormaDago

Result
The shortest distance between DagoSuitesApartment and BormaDago is 2.684 km
```

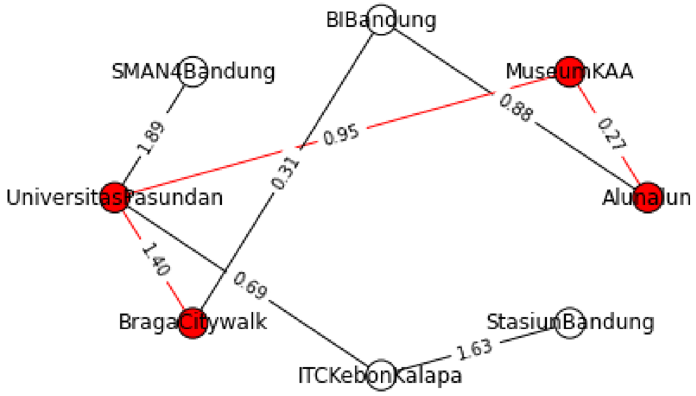


3.2. Screenshot Lintasan Terpendek di Daerah Alun-Alun Kota Bandung

Antara Alun-Alun dengan Braga City Walk

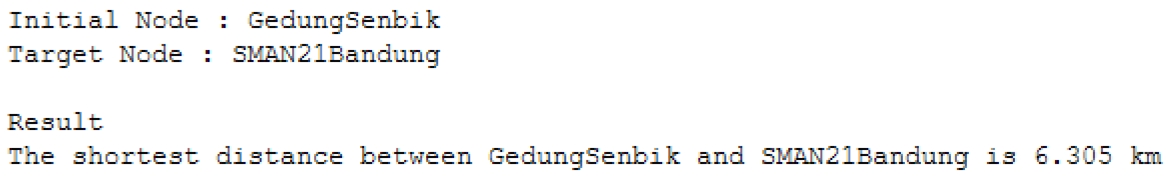


```
Initial Node : Alunalun
Target Node : BragaCitywalk

Result
The shortest distance between Alunalun and BragaCitywalk is 2.618 km
```

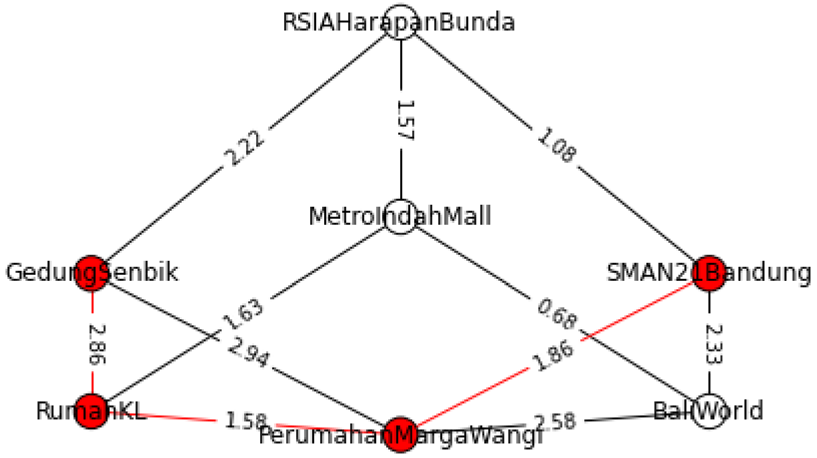


3.3. Screenshot Lintasan Terpendek di Daerah Buahbatu

Antara Gedung Senbik dengan SMAN 21 Bandung

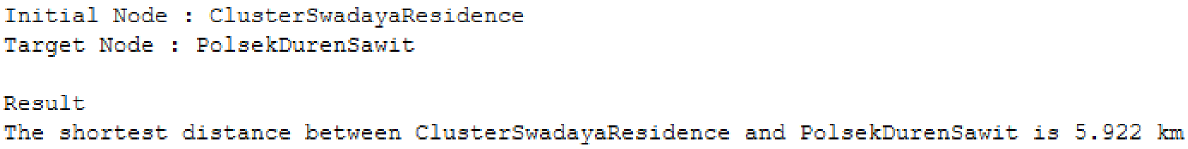


```
Initial Node : GedungSenbik
Target Node : SMAN21Bandung

Result
The shortest distance between GedungSenbik and SMAN21Bandung is 6.305 km
```

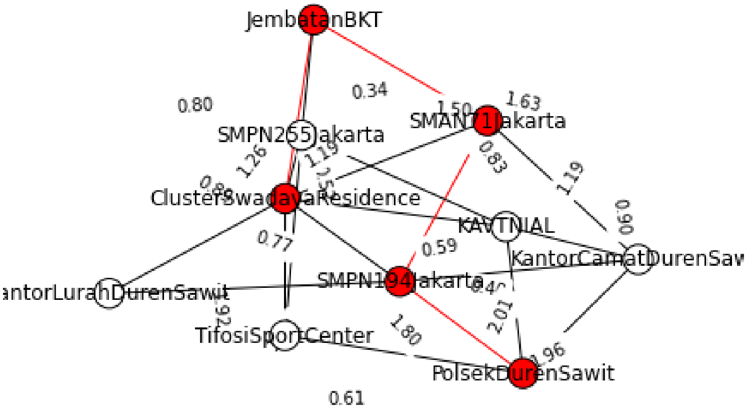


3.4. Screenshot Lintasan Terpendek di Daerah Duren Sawit

Antara Cluster Swadaya Residence dengan Polsek Duren Sawit

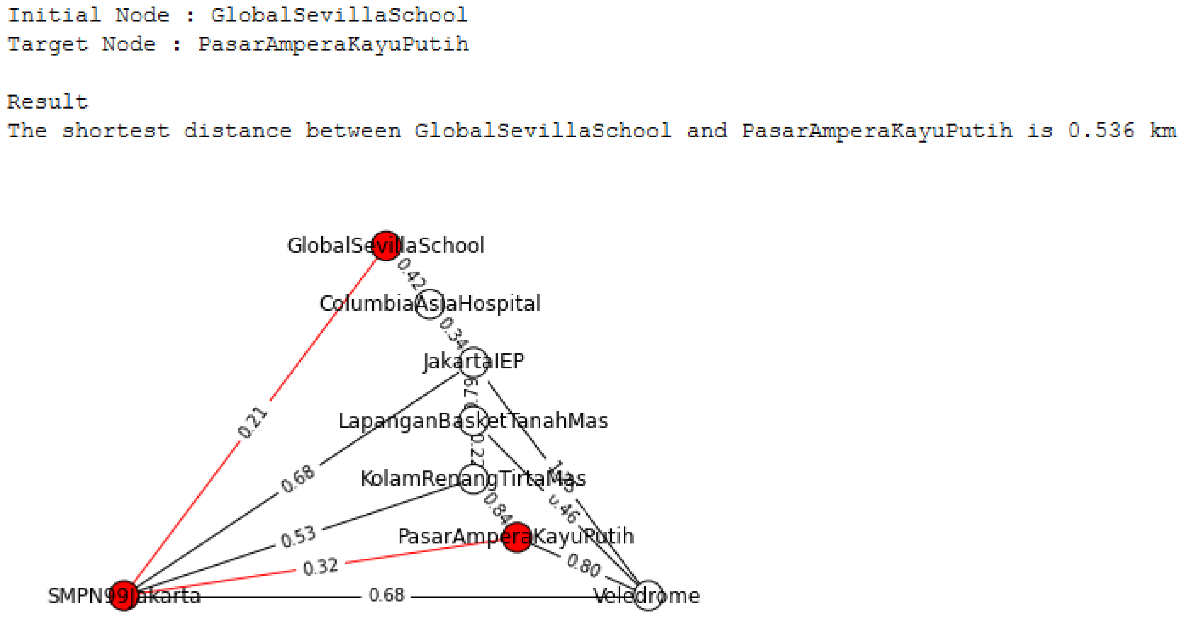


```
Initial Node : ClusterSwadayaResidence
Target Node : PolsekDurenSawit

Result
The shortest distance between ClusterSwadayaResidence and PolsekDurenSawit is 5.922 km
```



3.5. Screenshot Lintasan Terpendek di Daerah SMPN 99 Jakarta

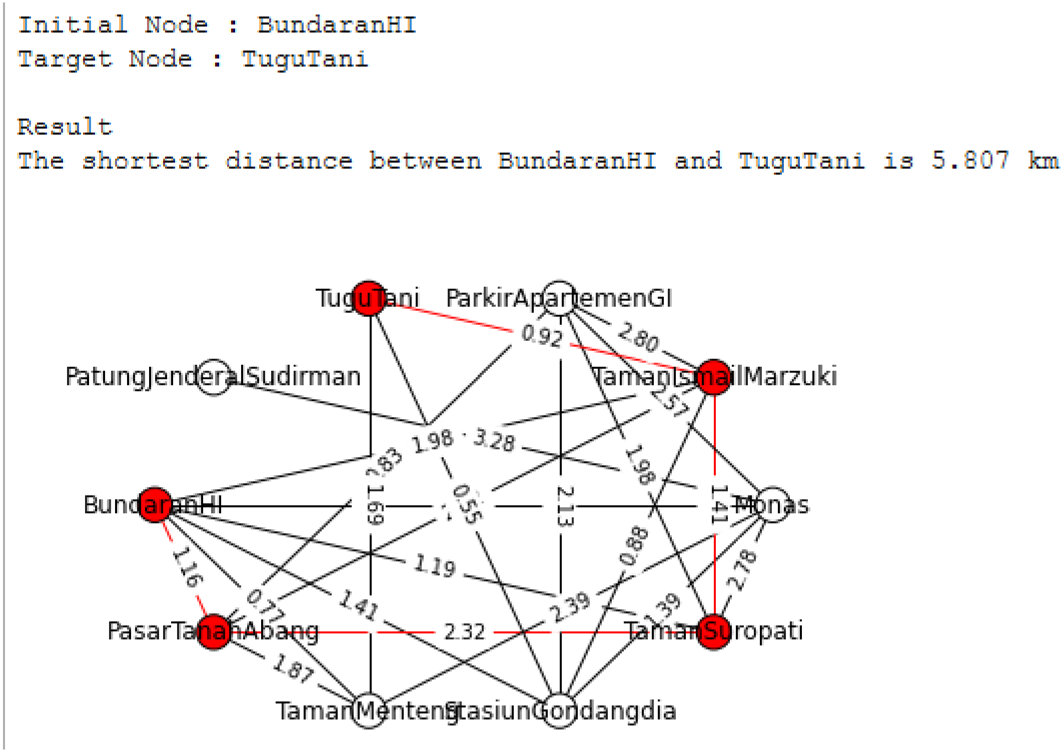
Antara Global Sevilla School dengan Pasar Ampera Kayu Putih



```
Initial Node : GlobalSevillaSchool
Target Node : PasarAmperaKayuPutih

Result
The shortest distance between GlobalSevillaSchool and PasarAmperaKayuPutih is 0.536 km
```

3.6. Screenshot Lintasan Terpendek di Daerah Bundaran HI

Antara Bundaran HI dengan Tugu Tani



```
Initial Node : BundaranHI
Target Node : TuguTani

Result
The shortest distance between BundaranHI and TuguTani is 5.807 km
```

IV. Link menuju kode program

GitHub

https://github.com/slarkdarr/Tucil3_13519107.git