DAT515 - Peer Review

Section 1: Core assignment

Q1: Yes
Q2: Yes
Q3: Yes

Section 2: Optional tasks

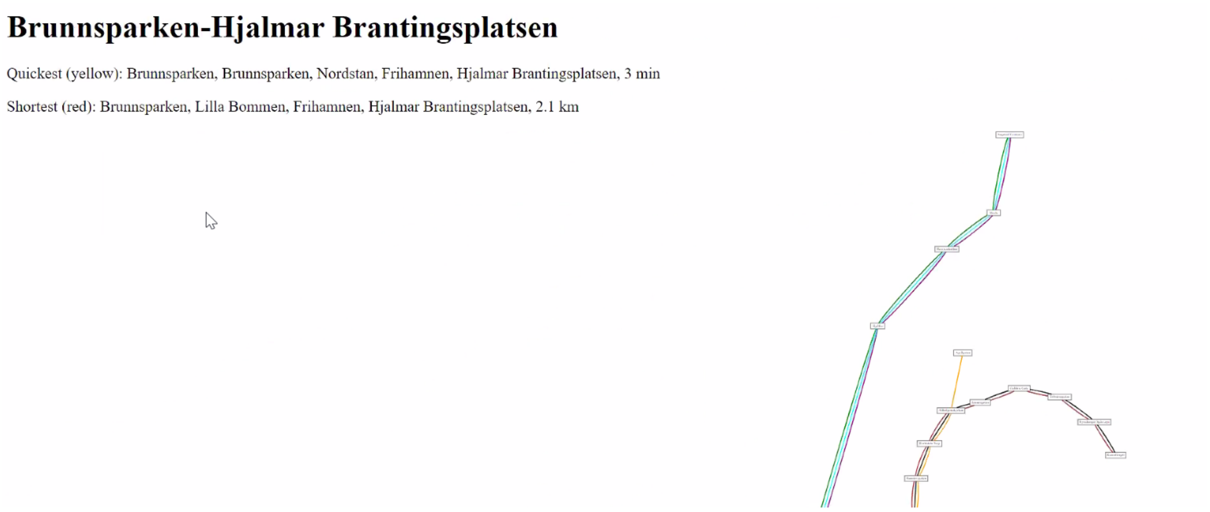
B1: Yes
B2: Yes

Section 3: Code Quality

Code has properly been reused from lab2 and only utilises one version of the dijkstra algorithm, as it should be. A suggestion might be to use the unitest library for testing all possible ways of running the code/giving inputs, in order to ensure no errors are hiding. Additionally, the code would’ve benefitted from some in-depth comments explaining more complex parts of the code leaving the reader with fewer abstraction layers to comprehend.

Section 4: Screenshots

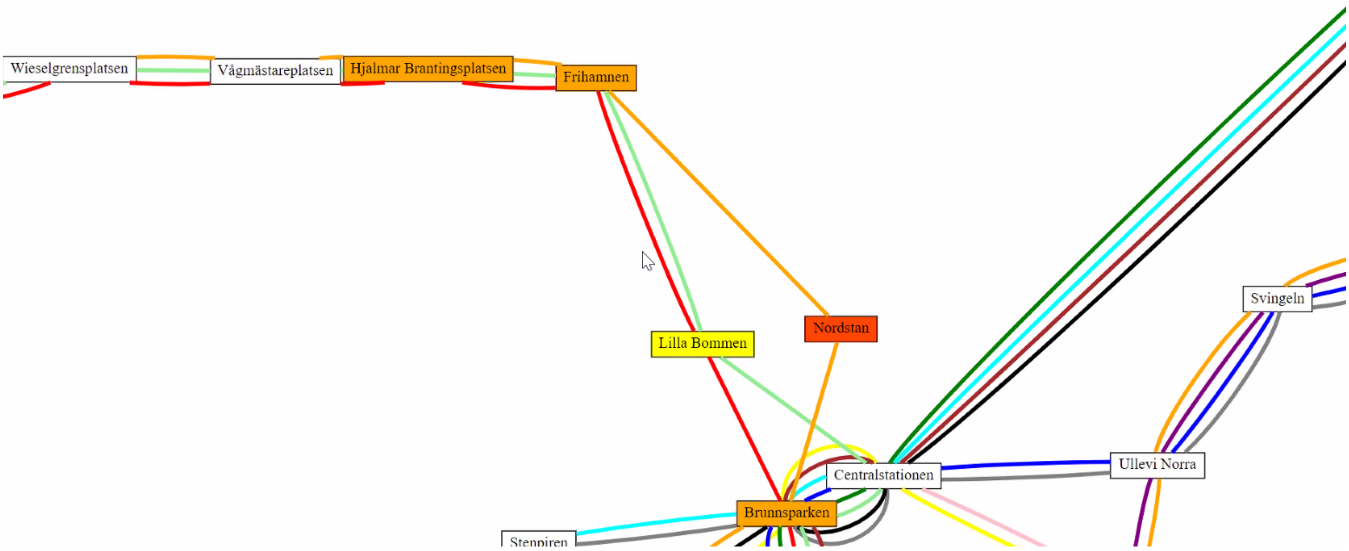
Screenshot_1



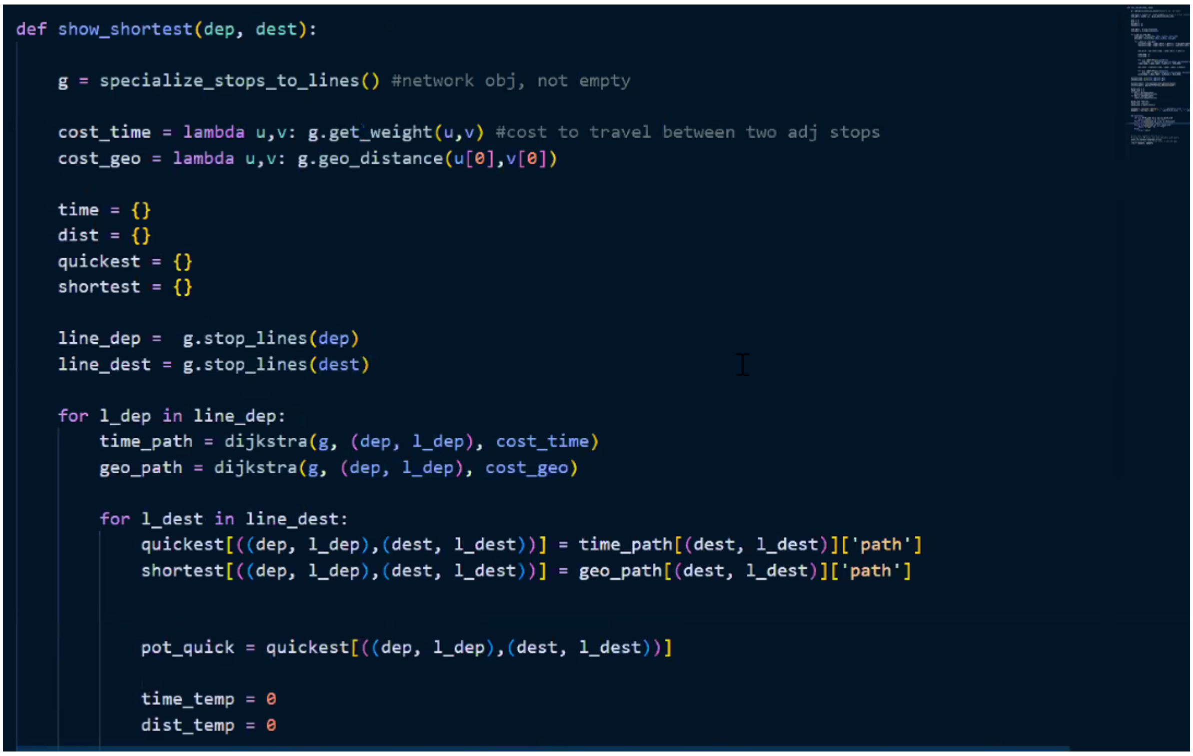
Brunnsparken-Hjalmar Brantingsplatsen

Quickest (yellow): Brunnsparken, Brunnsparken, Nordstan, Frihamnen, Hjalmar Brantingsplatsen, 3 min

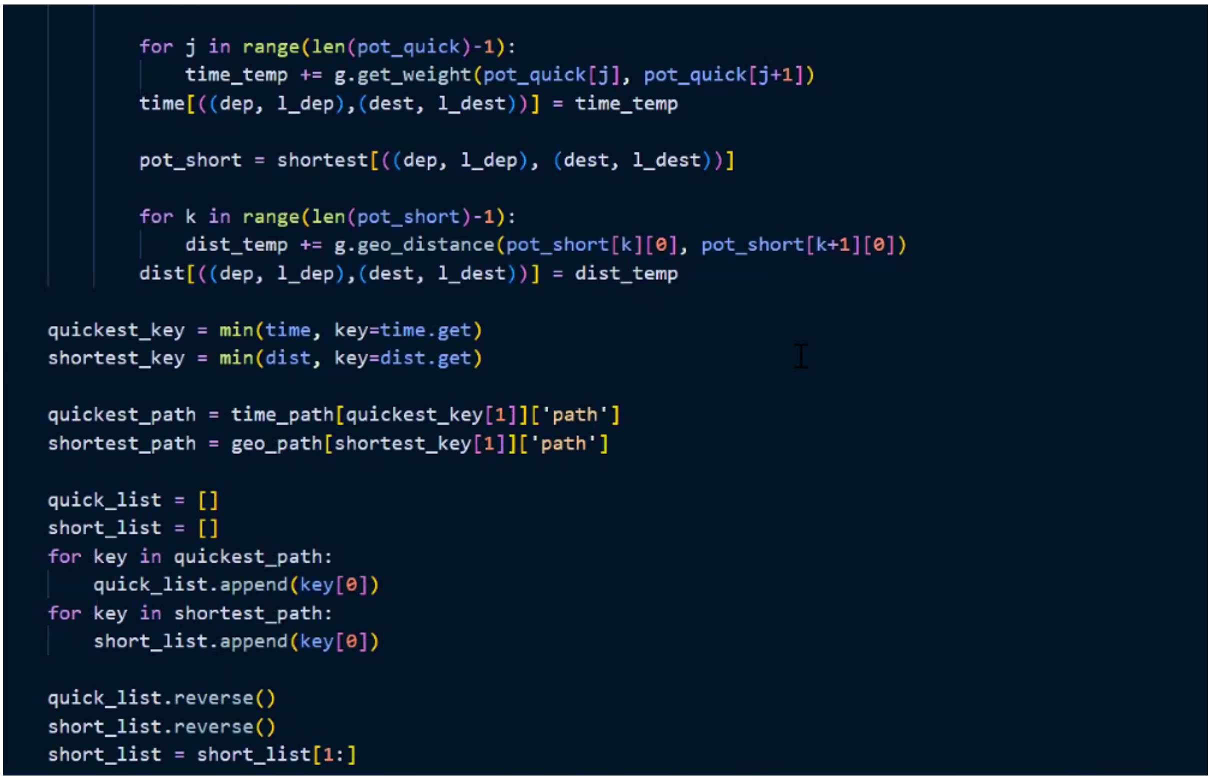
Shortest (red): Brunnsparken, Lilla Bommen, Frihamnen, Hjalmar Brantingsplatsen, 2.1 km


Wieselgrensplatsen
Vågmästareplatsen
Hjalmar Brantingsplatsen
Frihamnen
Svingeln
Lilla Bommen
Nordstan
Centralstationen
Ullevi Norra
Brunnsparken
Stenpiren

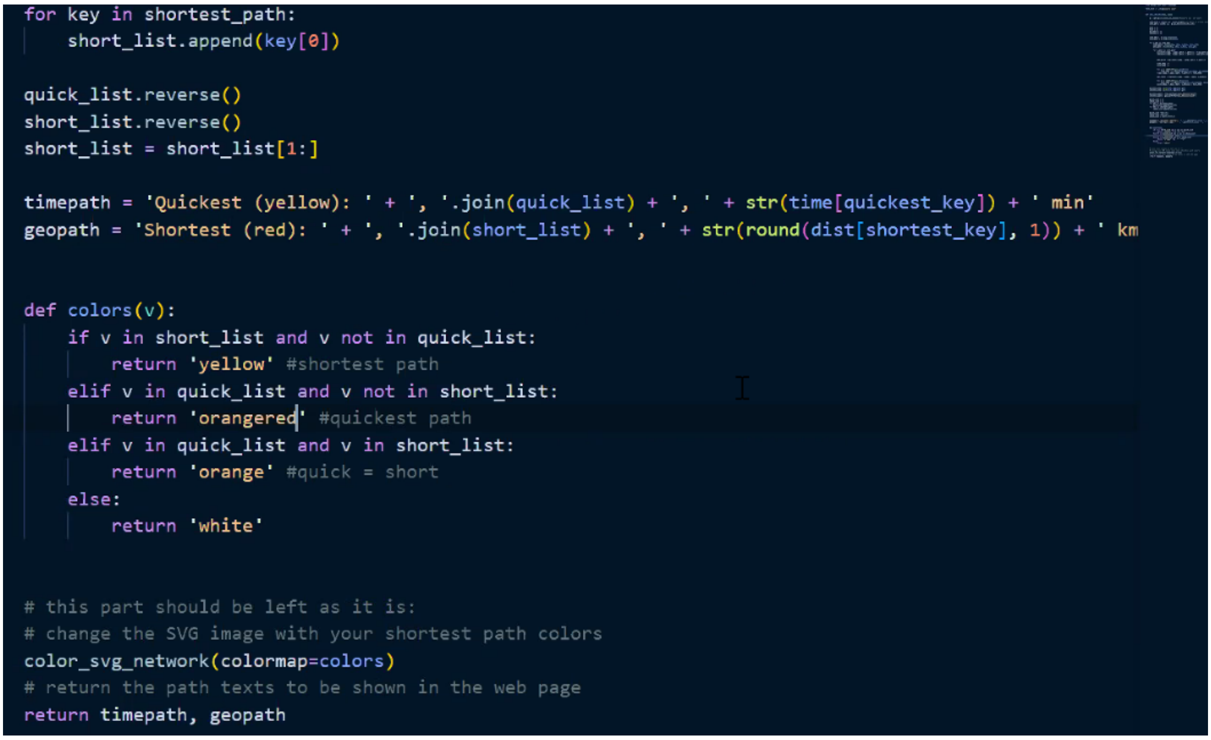
Screenshot_2



```
def show_shortest(dep, dest):

    g = specialize_stops_to_lines() #network obj, not empty

    cost_time = lambda u,v: g.get_weight(u,v) #cost to travel between two adj stops
    cost_geo = lambda u,v: g.geo_distance(u[0],v[0])

    time = {}
    dist = {}
    quickest = {}
    shortest = {}

    line_dep =  g.stop_lines(dep)
    line_dest = g.stop_lines(dest)

    for l_dep in line_dep:
        time_path = dijkstra(g, (dep, l_dep), cost_time)
        geo_path = dijkstra(g, (dep, l_dep), cost_geo)

        for l_dest in line_dest:
            quickest[((dep, l_dep),(dest, l_dest))] = time_path[(dest, l_dest)]['path']
            shortest[((dep, l_dep),(dest, l_dest))] = geo_path[(dest, l_dest)]['path']


            pot_quick = quickest[((dep, l_dep),(dest, l_dest))]

            time_temp = 0
            dist_temp = 0
```



```
            for j in range(len(pot_quick)-1):
                time_temp += g.get_weight(pot_quick[j], pot_quick[j+1])
            time[((dep, l_dep),(dest, l_dest))] = time_temp

            pot_short = shortest[((dep, l_dep), (dest, l_dest))]

            for k in range(len(pot_short)-1):
                dist_temp += g.geo_distance(pot_short[k][0], pot_short[k+1][0])
            dist[((dep, l_dep),(dest, l_dest))] = dist_temp

    quickest_key = min(time, key=time.get)
    shortest_key = min(dist, key=dist.get)

    quickest_path = time_path[quickest_key[1]]['path']
    shortest_path = geo_path[shortest_key[1]]['path']

    quick_list = []
    short_list = []
    for key in quickest_path:
        quick_list.append(key[0])
    for key in shortest_path:
        short_list.append(key[0])

    quick_list.reverse()
    short_list.reverse()
    short_list = short_list[1:]
```



```python
for key in shortest_path:
    short_list.append(key[0])

quick_list.reverse()
short_list.reverse()
short_list = short_list[1:]

timepath = 'Quickest (yellow): ' + ', '.join(quick_list) + ', ' + str(time[quickest_key]) + ' min'
geopath = 'Shortest (red): ' + ', '.join(short_list) + ', ' + str(round(dist[shortest_key], 1)) + ' km


def colors(v):
    if v in short_list and v not in quick_list:
        return 'yellow' #shortest path
    elif v in quick_list and v not in short_list:
        return 'orangered' #quickest path
    elif v in quick_list and v in short_list:
        return 'orange' #quick = short
    else:
        return 'white'


# this part should be left as it is:
# change the SVG image with your shortest path colors
color_svg_network(colormap=colors)
# return the path texts to be shown in the web page
return timepath, geopath
```