Peer review

DAT515

January 2023

Core assignment

Q1: Does the application run?
Yes

Q2: Does the application display the complete map of tram lines?
Yes

Q3: Is it possible to query shortest path between any two points?
Yes

Optional tasks

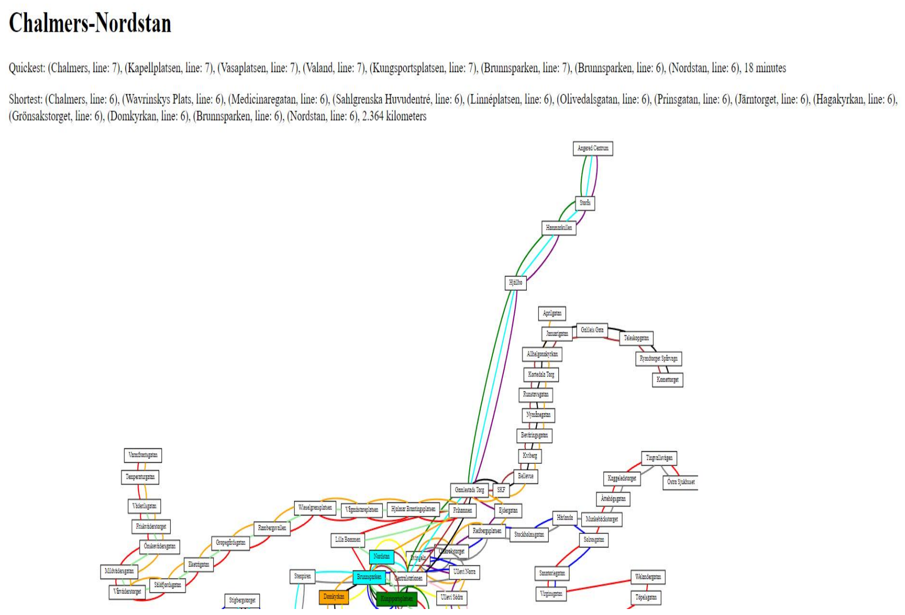
B1: Is the submission successfully accounting for Bonus Part 1?
Yes

B2: Is the submission successfully accounting for Bonus Part 2?
Yes

Code quality

Code quality is good. They implemented lab 2 as intended son no boilerplate code and used dijkstra efficiently, i.e. changes cost function instead of dijkstra function.

Screenshots



Chalmers-Nordstan

Quickest: (Chalmers, line: 7), (Kapellplatsen, line: 7), (Vasaplatsen, line: 7), (Valand, line: 7), (Kungsportsplatsen, line: 7), (Brunnsparken, line: 7), (Brunnsparken, line: 6), (Nordstan, line: 6), 18 minutes

Shortest: (Chalmers, line: 6), (Wavrinskys Plats, line: 6), (Medicinaregatan, line: 6), (Sahlgrenska Huvudentré, line: 6), (Linnéplatsen, line: 6), (Olivedalsgatan, line: 6), (Prinsgatan, line: 6), (Järntorget, line: 6), (Hagakyrkan, line: 6), (Grönsakstorget, line: 6), (Domkyrkan, line: 6), (Brunnsparken, line: 6), (Nordstan, line: 6), 2.364 kilometers

Figure 1: Screenshot 1

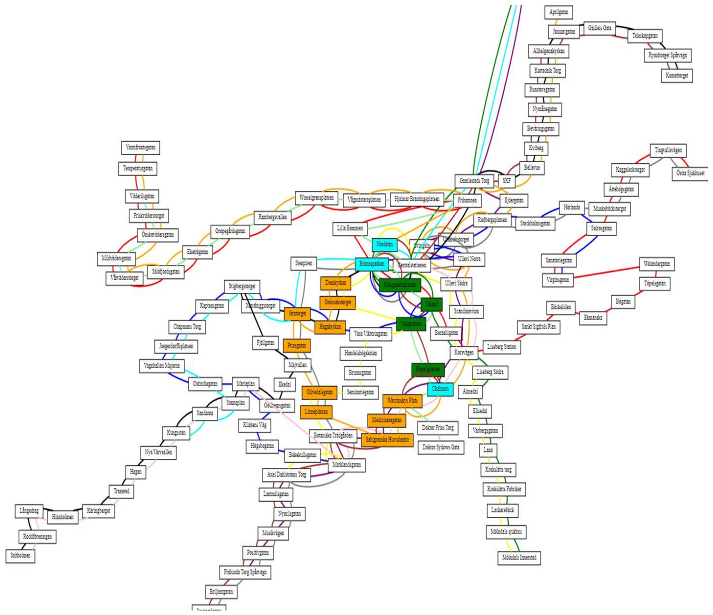
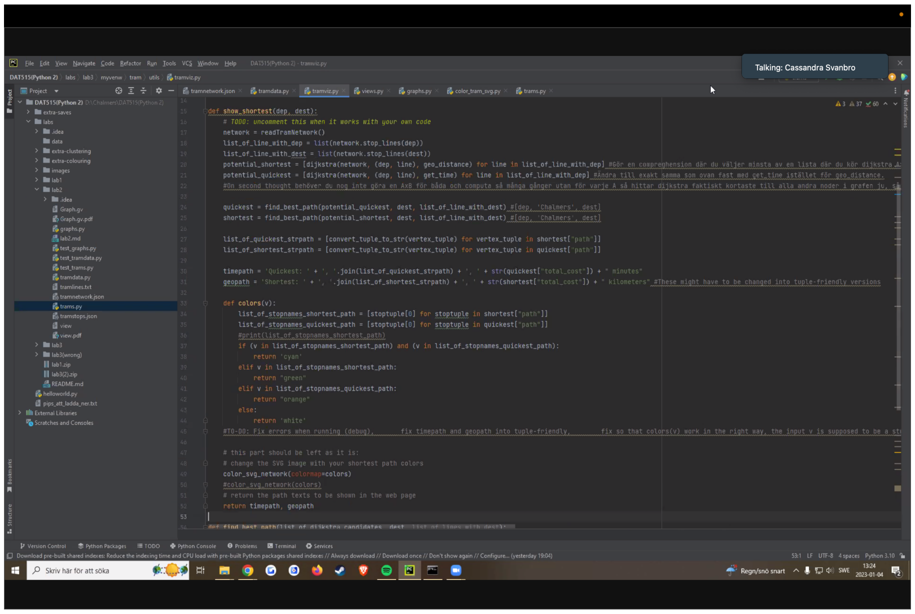
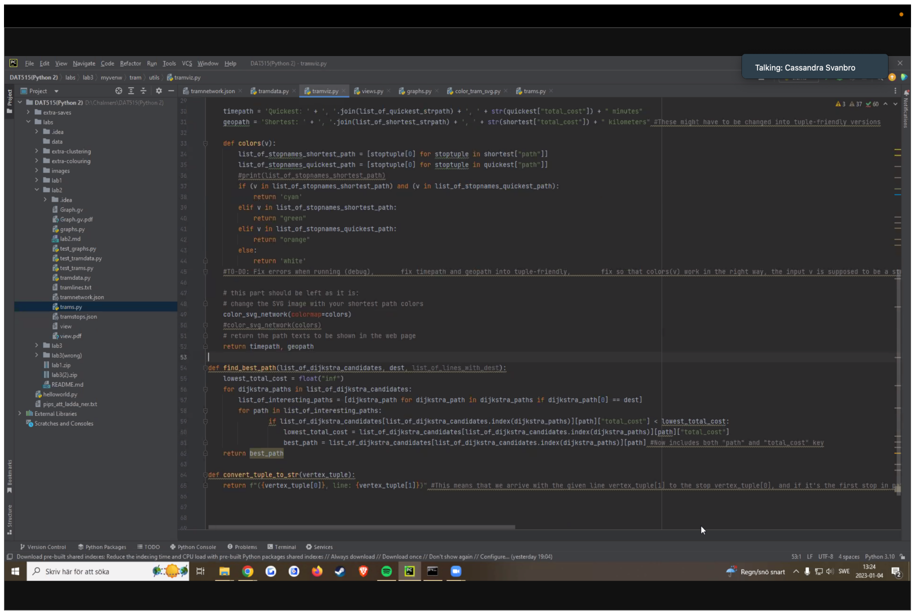


Figure 2: Screenshot 1



```python
def show_shortest(dep, dest):
    # TODO: uncomment this when it works with your own code
    network = readTramNetwork()
    list_of_line_with_dep = list(network.stop_lines(dep))
    list_of_line_with_dest = list(network.stop_lines(dest))
    potential_shortest = [dijkstra(network, (dep, line), geo_distance) for line in list_of_line_with_dep] #Gör en compreghension där du väljer minsta av en lista där du kör dijkstra A
    potential_quickest = [dijkstra(network, (dep, line), get_time) for line in list_of_line_with_dep] #Ändra till exakt samma som ovan fast med get_tine istället för geo_distance.
    #On second thought behöver du nog inte göra en AxB för båda och computa så många gånger utan för varje A så hittar dijkstra faktiskt kortaste till alla andra noder i grafen ju, så

    quickest = find_best_path(potential_quickest, dest, list_of_line_with_dest) #[dep, 'Chalmers', dest]
    shortest = find_best_path(potential_shortest, dest, list_of_line_with_dest) #[dep, 'Chalmers', dest]

    list_of_quickest_strpath = [convert_tuple_to_str(vertex_tuple) for vertex_tuple in shortest["path"]]
    list_of_shortest_strpath = [convert_tuple_to_str(vertex_tuple) for vertex_tuple in quickest["path"]]

    timepath = 'Quickest: ' + ', '.join(list_of_quickest_strpath) + ', ' + str(quickest["total_cost"]) + " minutes"
    geopath = 'Shortest: ' + ', '.join(list_of_shortest_strpath) + ', ' + str(shortest["total_cost"]) + " kilometers" #These might have to be changed into tuple-friendly versions

    def colors(v):
        list_of_stopnames_shortest_path = [stoptuple[0] for stoptuple in shortest["path"]]
        list_of_stopnames_quickest_path = [stoptuple[0] for stoptuple in quickest["path"]]
        #print(list_of_stopnames_shortest_path)
        if (v in list_of_stopnames_shortest_path) and (v in list_of_stopnames_quickest_path):
            return 'cyan'
        elif v in list_of_stopnames_shortest_path:
            return "green"
        elif v in list_of_stopnames_quickest_path:
            return "orange"
        else:
            return 'white'
    #TO-DO: Fix errors when running (debug),       fix timepath and geopath into tuple-friendly,       fix so that colors(v) work in the right way, the input v is supposed to be a st

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

def find_best_path(list_of_dijkstra_candidates, dest, list_of_lines_with_dest):
```

Figure 3: Screenshot 2



```python
    timepath = 'Quickest: ' + ', '.join(list_of_quickest_strpath) + ', ' + str(quickest["total_cost"]) + " minutes"
    geopath = 'Shortest: ' + ', '.join(list_of_shortest_strpath) + ', ' + str(shortest["total_cost"]) + " kilometers" #These might have to be changed into tuple-friendly versions

    def colors(v):
        list_of_stopnames_shortest_path = [stoptuple[0] for stoptuple in shortest["path"]]
        list_of_stopnames_quickest_path = [stoptuple[0] for stoptuple in quickest["path"]]
        #print(list_of_stopnames_shortest_path)
        if (v in list_of_stopnames_shortest_path) and (v in list_of_stopnames_quickest_path):
            return 'cyan'
        elif v in list_of_stopnames_shortest_path:
            return "green"
        elif v in list_of_stopnames_quickest_path:
            return "orange"
        else:
            return 'white'
    #TO-DO: Fix errors when running (debug),       fix timepath and geopath into tuple-friendly,       fix so that colors(v) work in the right way, the input v is supposed to be a st

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

def find_best_path(list_of_dijkstra_candidates, dest, list_of_lines_with_dest):
    lowest_total_cost = float("inf")
    for dijkstra_paths in list_of_dijkstra_candidates:
        list_of_interesting_paths = [dijkstra_path for dijkstra_path in dijkstra_paths if dijkstra_path[0] == dest]
        for path in list_of_interesting_paths:
                if list_of_dijkstra_candidates[list_of_dijkstra_candidates.index(dijkstra_paths)][path]["total_cost"] < lowest_total_cost:
                    lowest_total_cost = list_of_dijkstra_candidates[list_of_dijkstra_candidates.index(dijkstra_paths)][path]["total_cost"]
                    best_path = list_of_dijkstra_candidates[list_of_dijkstra_candidates.index(dijkstra_paths)][path] #Now includes both "path" and "total_cost" key
    return best_path

def convert_tuple_to_str(vertex_tuple):
    return f"({vertex_tuple[0]}, line: {vertex_tuple[1]})" #This means that we arrive with the given line vertex_tuple[1] to the stop vertex_tuple[0], and if it's the first stop in p
```

Figure 4: Screenshot 2