Ali Alkhaled reporting Arvid Imgård, Edvin Gustavsson

Section 1: 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**

Section 2: Optional tasks

- B1: Is the submission successfully accounting for Bonus Part 1? --> **Yes**
- B2: Is the submission successfully accounting for Bonus Part 2? --> **Yes**

Section 3: Code quality

The code seems to be well structured where the code from previous lab parts has been reused. When testing the application, the search feature does not take unnecessarily long time to show the result, which also indicates that the code does not contain any unnecessary overcomplex logic, such as too many (nested)loops. Some code documentation, however, would be good to have in order to ease the development process in the future projects, especially when collaborating with many developers.

- code from lab 2 has been properly reused (i.e., in an efficient way without boilerplate code)

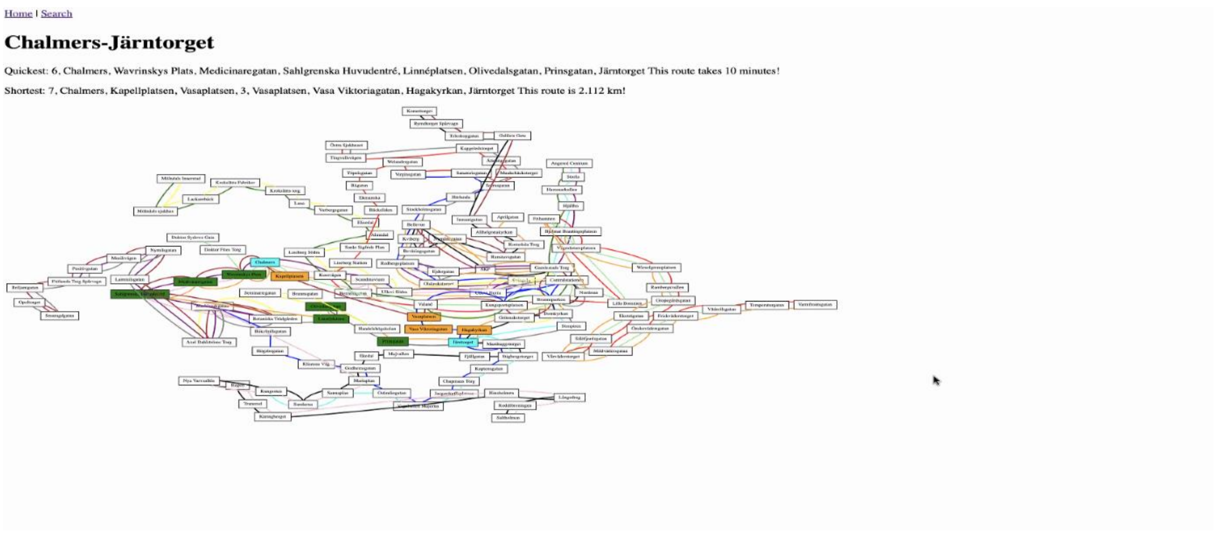
  The answer is: Yes!

- dijkstra has been implemented and used as intended: there is just one definition of the function itself, and different distances are obtained by just changing the cost function

  The answer is: Yes!

Section 4: Screenshots

Screenshot 1



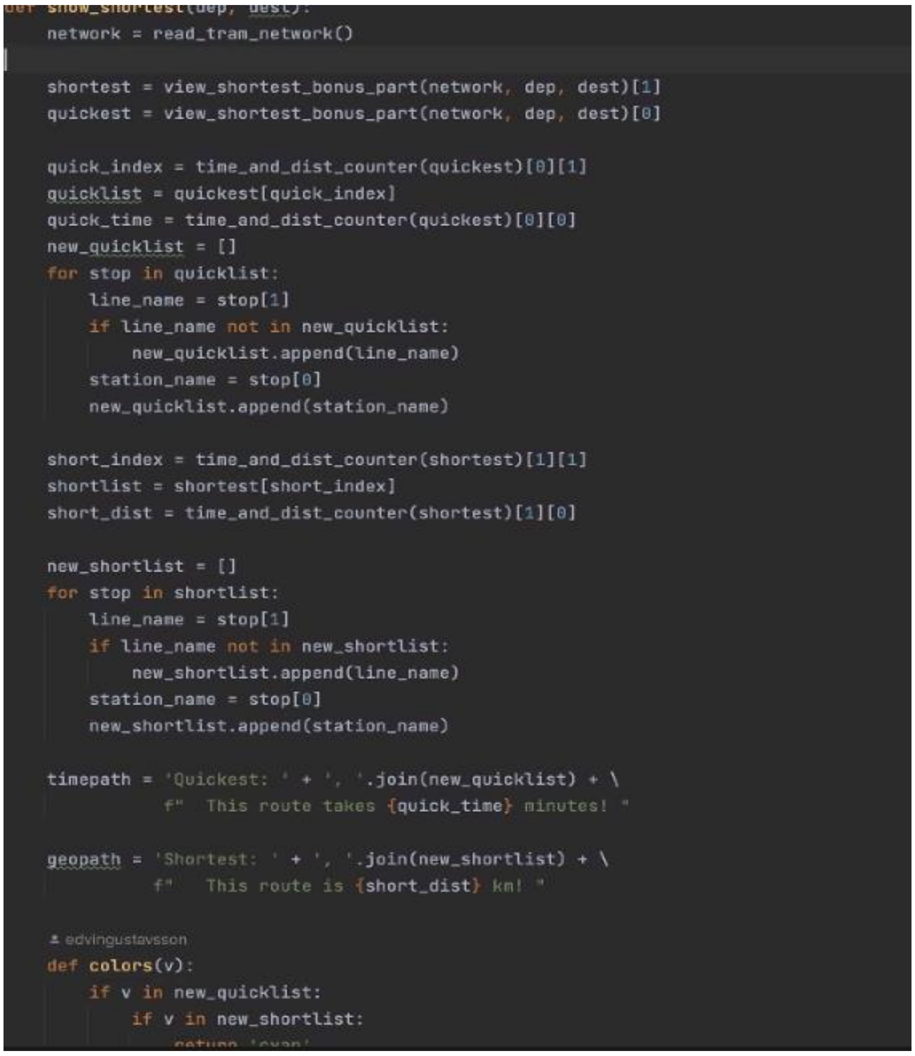
Home | Search

Chalmers-Järntorget

Quickest: 6, Chalmers, Wavrinskys Plats, Medicinaregatan, Sahlgrenska Huvudentré, Linnéplatsen, Olivedalsgatan, Prinsgatan, Järntorget This route takes 10 minutes!

Shortest: 7, Chalmers, Kapellplatsen, Vasaplatsen, 3, Vasaplatsen, Vasa Viktoriagatan, Hagakyrkan, Järntorget This route is 2.112 km!

Screenshot 2



```
def show_shortest(dep, dest):
    network = read_tram_network()

    shortest = view_shortest_bonus_part(network, dep, dest)[1]
    quickest = view_shortest_bonus_part(network, dep, dest)[0]

    quick_index = time_and_dist_counter(quickest)[0][1]
    quicklist = quickest[quick_index]
    quick_time = time_and_dist_counter(quickest)[0][0]
    new_quicklist = []
    for stop in quicklist:
        line_name = stop[1]
        if line_name not in new_quicklist:
            new_quicklist.append(line_name)
        station_name = stop[0]
        new_quicklist.append(station_name)

    short_index = time_and_dist_counter(shortest)[1][1]
    shortlist = shortest[short_index]
    short_dist = time_and_dist_counter(shortest)[1][0]

    new_shortlist = []
    for stop in shortlist:
        line_name = stop[1]
        if line_name not in new_shortlist:
            new_shortlist.append(line_name)
        station_name = stop[0]
        new_shortlist.append(station_name)

    timepath = 'Quickest: ' + ', '.join(new_quicklist) + \
               f"  This route takes {quick_time} minutes! "

    geopath = 'Shortest: ' + ', '.join(new_shortlist) + \
              f"  This route is {short_dist} km! "

    ≗ edvingustavsson
    def colors(v):
        if v in new_quicklist:
            if v in new_shortlist:
                return 'cyan'
```