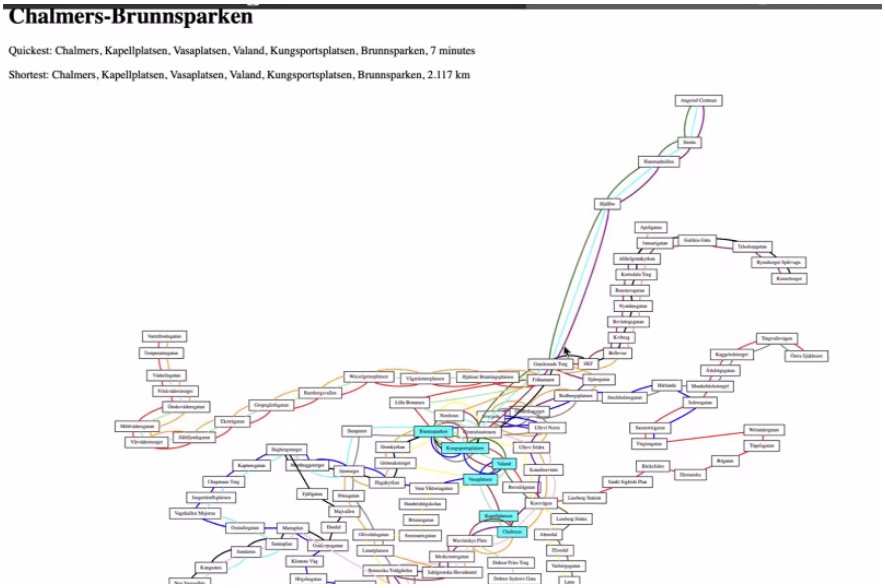
**Peer review, Ludvig Lehmann reviewing Tony Gromer**

The application runs and displays the map of tramlines. Using the search function it is possible to find the shortest path between any two stops. Both bonus tasks had been completed; the application accounts for changing lines when computing travel times and the map links to actual time tables.

Overall the code was well structured and easily readable. The code from lab 2 was reused efficiently without unnecessary repetition. The Dijkstra implementation also functioned as intended; it was one function that, given a cost function, computed shortest paths and distances from a given stop.

**Screenshot 1:**




**Screenshot 2:**

DAT515 > DAT515_Tony_Gromer > lab3 > tram > utils > tramviz.py > show_shortest

```
    if Bonus1:
        network.specialize_stops_to_lines()

        dep_list = [stop for stop in network.vertices() if stop[0] == dep]
        dest_list = [stop for stop in network.vertices() if stop[0] == dest]
        paths_time = []
        paths_dist = []

        for stop1 in dep_list:
            time_dij = dijkstra(network, stop1, network.specialized_transition_time)
            dist_dij = dijkstra(network, stop1, network.specialized_geo_distance)

            for stop2 in dest_list:
                paths_time.append(time_dij[stop2])
                paths_dist.append(dist_dij[stop2])

        quickest_nodes = min(paths_time, key = lambda x: x['dist'])
        shortest_nodes = min(paths_dist, key = lambda x:x['dist'])

        quickest = [stop[0] for stop in quickest_nodes['path']]
        shortest = [stop[0] for stop in shortest_nodes['path']]

    else:
        quickest_nodes = dijkstra(network,dep, network.transition_time)[dest]
        shortest_nodes = dijkstra(network, dep, network.geo_distance)[dest]

        quickest = quickest_nodes['path']
        shortest = shortest_nodes['path']

    timepath = 'Quickest: ' + ', '.join(quickest) + ', ' + str(quickest_nodes['dist']) + ' minutes'
    geopath = 'Shortest: ' + ', '.join(shortest) + ', ' + str(round(shortest_nodes['dist'],3)) + ' km'

    def colors(v):
        if v in shortest and v in quickest:
            return 'cyan'
        elif v in quickest:
            return 'orange'
        elif v in shortest:
            return 'green'
```

**Grade:** Complete