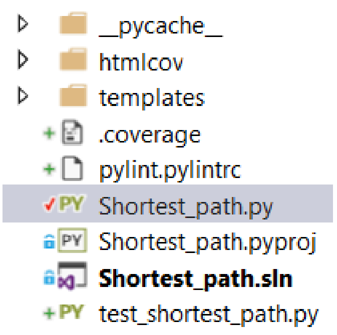
Toteutusdokumentti _ Implementation document

Data Structures Lab Course (start of summer 2023)

Structure of the program:

The current files are structured as follows in Visual Studio. Shortest_path.py is the application file which contains all the codes for the application, incl. user interface, various functions and route. Because this project requires comparatively small amount of functions and methods, that's why didn't split functions into different files. The last file test_shortest_path.py is the test case file.



```
_pycache_
htmlcov
templates
.coverage
pylint.pylintrc
Shortest_path.py
Shortest_path.pyproj
Shortest_path.sln
test_shortest_path.py
```

Processes to implement the program:

- Import the necessary libraries: Flask, folium, polyline, geopy.geocoders.Nominatim etc.
- Create a Flask application instance and html file as the user interface.
- Define the geocoding function (geocode) that takes an address as input, and returns corresponding coordinates using the Nominatim geocoding service.
- Define the route handling for the home page ("/") that handles both GET and POST requests. In the GET request, render the initial index.html template. In the POST request, retrieve the start and end addresses from the user interface's form, geocode the addresses, and proceed with finding the shortest path.
- Implement the find_shortest_path function with IDA* algorithm, which is the core and target for this course. This function sends a request to the external API to calculate the driving route between the start and end coordinates. It retrieves the encoded polyline from the response and decodes it using polyline.decode.
- Implement the plot_shortest_path function, which creates a Folium map, adds markers for the start and end coordinates, and plots the polyline representing the shortest path. The map is saved as an HTML file, which will be rendered in browser.
- Run this Flask application using app.run(). Users can test the program by entering start and end addresses, submitting the form, and verifying that the map with the shortest path is displayed.

Time and space complexities:

Because this project uses geocoding and HTTP requests, complexities can be impacted by these both actions. It requires space to store the addresses and coordinates. The biggest impact would be the path finding algorithm, in the current codes networkx library with built-in Dijkstra's algorithm bring the time complexity of $O((V + E) \log V)$, V is the number of nodes, and E is the number of edges. The space complexity is $O(V + E)$. In the last step of plotting, the amount of the polyline and nodes will impact on the time complexity, and space complexity is decided by the amount of the coordinates and the map file.

Performance:

If the internet connection slows down or the map API works inefficiently, it will impact on how quick geocoding and html file can be executed.

Possible flaws and improvements:

In order to display the updated html file correctly, codes were modified to allow generate new html file to local folder for new address's submission, which is not an ideal solution if considering the huge memory demand for saving those html files. Thus, this current solution is not for a big amount of users. From another perspective, the advantage is service administrator is able to record and retrieve all the path finding records. As an alternative, in real life the service administrator can remove or delete those html files with a certain frequency if those are not needed for user management. Till now, path results visualization is successful.

Sources:

Flask: Quickstart — Flask Documentation (2.3.x) (palletsprojects.com)

Flask Tutorial in Visual Studio Code: https://code.visualstudio.com/docs/python/tutorial-flask