# OTIV software challenge

9 November 2023

## 1 Introduction

Hello, potential OTIV colleague! If you are reading this, you are applying for a software developer job at OTIV. We would like you to solve this challenge in order to evaluate your skills and see whether you're a good fit for the team. Given a 2D rail map and a list of random target positions, the goal is to write a C++ or Rust program that locates the closest point on the rail map for each target position. Please read the full instructions before starting this assignment!

## 2 Assignment details

### 2.1 Inputs

You are given the tram rail network in Antwerp which is described in `inputs/rail_map.txt`. A rail map is defined as an undirected graph which consists of nodes and edges. A node represents a 2D location on the map while an edge represents a rail track connecting two nodes. A node has three properties:

- `node_id (integer)`: the node's unique ID which is a number between 0 and $N - 1$ where $N$ is the total number of nodes.
- `lat (float)`: the latitude of this node in degrees.
- `lon (float)`: the longitude of this node in degrees.

An edge connects two nodes and therefore has two properties:

- `id0 (integer)`: the id of the first node of the edge
- `id1 (integer)`: the id of the second node of the edge

Since the graph is undirected, the order of the `id0` and `id1` does not matter. The nodes and edges can be extracted from `inputs/rail_map.txt` , for more details check the rail map file format in appendix 5.3.

You are also given a list of target positions in Antwerp. These locations are randomly spread across the city, they are not part of the rail map! Each target location has these properties:

- `target_id (integer)`: the target's unique ID which is a number between 0 and $T-1$ where $T$ is the total number of targets.
- `lat (float)`: the latitude of this target in degrees.
- `lon (float)`: the longitude of this target in degrees.

The target positions can be extracted from `inputs/targets.txt` , for more details check the targets file format in appendix 5.3.

## 2.2 Goals

The goal of this assignment is to find for each target the closest point on the rail map. A point on the rail map is defined as a point that belongs to one of the rail map's **edges** (!). An edge is a straight line between two nodes and each point on this line is considered to be part of the rail map. So it's not sufficient to find the node closest to the target! The solution is always a point located on the line represented by an edge.

Since each point is defined by its latitude and longitude, it is not straightforward to find the distance between two given points. Therefore do the following: for each point convert the latitude and longitude $(\phi, \psi)$ to an $(x, y)$ coordinate in meters according to appendix 5.2. All distances must be calculated using the $(x, y)$ coordinates of each point or line. Also note that in this assignment we always assume **2D Euclidean** distances when mentioning distances.

Each solution for a given target has the following properties:

- `target_id (integer)`: the target's unique ID which is a number between 0 and $T-1$ where $T$ is the total number of targets/solutions.
- `id0 (integer)`: the first node ID of the edge containing the closest point to the target.
- `id1 (integer)`: the second node ID of the edge containing the closest point to the target.
- `lat (float)`: the latitude of the closest point in degrees.
- `lon (float)`: the longitude of the closest point in degrees.
- `distance (float)`: the 2d Euclidean distance (in meters) between the closest point and the target point. This number is the metric that should be minimised in this assignment.

The assignment consists of two parts:

1. Write a C++ **or** Rust program that does the following steps:
    - Parse the input files located in `inputs/`.

    - For each target, find the closest point on the map.
    - Write these solutions to a new file called `outputs/solutions.txt`. The format for this solutions files can be found in appendix 5.3.
    - **Important**: don't use any external libraries, only the standard library is allowed to be used.

2. Create a Python Jupyter notebook `visuals.ipynb` which visualizes the rail map, the target points and the solutions. We require you to use the Folium (`https://python-visualization.github.io/folium/latest/`) library to achieve this. This library allows to draw lines and points on 2D maps using latitudes and longitudes as inputs. The notebook should do the following steps:
    - Parse the input files (`inputs/*.txt`) and solutions file (`outputs/solutions.txt`) which you generated in the previous step.
    - Visualise the rail map, the target points and solutions in one Folium interactive map (see example in `visuals.ipynb`) .

## 3 Deliverables

To deliver this assignment back to OTIV, zip this folder and send it over mail to `pieter@otiv.ai`. Make sure it contains the following parts:

- All C++ source files if you used C++ for part 1.
- All Rust sources files if you used Rust for part 1.
- The given input files `inputs/*.txt`.
- The solutions file `outputs/solutions.txt` generated in part 1.
- The Jupyter notebook `visuals.ipynb` containing all code and the visual for part 2.

## 4 Presentation

Defend your assigment during a scheduled meeting with one of the OTIV members. The OTIV member will ask detailed questions about the delivered solution. It's important to deliver **clean code** which is easy to read!

## 5 Appendix

### 5.1 Given templates

For part 1 we recommend to start from the given files in this directory.

- **C++**: the main file is `src/main.cpp`. We also provided a sample `CMakeLists.txt` so compiling and running can be done like this:

```
    cmake -S . -B build/
    cmake --build build/
    ./build/main

```

  Feel free to modify these files and add additional source files.

- **Rust**: the main file is `src/main.rs`. We also provided a sample `Cargo.toml` so compiling and running can be done like this:

```
    cargo run

```

  Feel free to modify these files and add additional source files.

For part 2 you can fill in the given `visuals.ipynb` file. It contains an example on how to display an interactive map in the notebook itself. You only need to add code to read and draw the rail map, targets and solutions to that map.

## 5.2 Conversion to $xy$ coordinates

All points on the map are given in latitude-longitude $(\phi, \psi)$ coordinates. You first need to convert these points to cartesian $(x, y)$ coordinates in meter. For that you need to use the reference latitude and longitude $(\phi_{ref}, \psi_{ref})$ which are given in the rail map file. This reference points correspond to the point where $(x, y) = (0, 0)$.

To convert any latitude-longitude coordinates $(\phi, \psi)$ (in degrees) to cartesian coordinates $(x, y)$ (in meters) and vice versa, use the following formulas:

$$\begin{aligned}
\Delta_\psi &= 111412.84\cos(\phi_{ref}) - 93.5\cos(3\phi_{ref}) + 0.118\cos(5\phi_{ref}) \\
\Delta_\phi &= 111132.92 - 559.82\cos(2\phi_{ref}) + 1.175\cos(4\phi_{ref}) - 0.0023\cos(6\phi_{ref}) \\
x &= \Delta_\psi * (\psi - \psi_{ref}) \\
y &= \Delta_\phi * (\phi - \phi_{ref}) \\
\psi &= \psi_{ref} + \frac{x}{\Delta_\psi} \\
\phi &= \phi_{ref} + \frac{y}{\Delta_\phi}
\end{aligned}$$

## 5.3 File formats

This guide describes the input and output txt file formats. Use this guide for parsing input files and writing the solution output file.

- `inputs/rail_map.txt`:

```
        reference <lat_ref> <lon_ref>
        nodes <N>
        0 <lat> <lon>
        1 <lat> <lon>
        ...
        N-1 <lat> <lon>
        edges <E>
        <id0> <id1>
        <id0> <id1>
        ...
        <id0> <id1>

```

The first line is `reference <lat_ref> <lon_ref>` which gives the reference latitude and longitude used to convert lat/lon tuples to $xy$ coordinates. The second line is `nodes <N>` where `N` is the number of nodes of the rail map. Next follow `N` lines where each line represents a node. These lines have the format `<node_id> <lat> <lon>` which correspond to the nodes' ID, latitude and longitude respectively. The next line is `edges <E>` where `E` is the number of edges of the rail map. Next follow `E` lines where each line represents an edge. These lines have the format `<id0> <id1>` which refer to the IDs of the two nodes that are connected by this edge.

- `inputs/targets.txt`:

```
targets <T>
0 <lat> <lon>
1 <lat> <lon>
...
T-1 <lat> <lon>
```

The first line is `targets <T>` where `T` is the number of targets. Next follow `T` lines where each line represents a target. These lines have the format `<target_id> <lat> <lon>` which correspond to the target's ID, latitude and longitude respectively.

- `outputs/solutions.txt`:

```
solutions <T>
0 <id0> <id1> <lat> <lon> <distance>
1 <id0> <id1> <lat> <lon> <distance>
...
T-1 <id0> <id1> <lat> <lon> <distance>
```

The first line is solutions `T` where `T` is the number of solutions (which is equal to the number of targets). Next follow `T` lines where each line represents a solution. These lines have the format `<target_id> <id0> <id1> <lat> <lon> <distance>` which correspond to the solution properties explained in the instructions.