Code Challenge - Shortest Paths on OpenStreetMap

This challenge menu contains three parts touching the three layers respectively of a typical geospatial information system: data processing, analytics, and visualization. The major ingredient for this "meal" is OpenStreetMap, OSM data. In brief, you will implement the shortest path algorithm that can calculate the optimal **pedestrian** path between a `<source, target>` pair from OSM road networks.

🍹 Appetizer: data preparation

First of all, it's required to parse OSM data into an appropriate structure that the routing algorithm can work with. To parse/clean the raw OSM data is not a trivial task. So it's encouraged to utilize third-party libraries or tools to help you. There is a long list of open-source projects that can process the raw data. Some of them can even do network analysis, e.g. OSMnx, OSRM. Please feel free to choose your favorite one.

Besides, to reduce the resource consumption and standardize the expected results, please use the OSM data within bbox `[11.5430,48.1249,11.6104,48.1510]` as the testbed covering the central area of Munich city. We have already exported the raw OSM data within the above bbox and zipped it in the attachment for convenience. But you can surely fetch the data in other ways by yourself. It is worth noting that the number of street links eligible for pedestrians will vary depending on the applied filters. So please select the reasonable ones in your opinion.

🍜 Main course: shortest path algorithm *Two-Q* implementation

In this second part, you will need to implement an algorithm proposed by Pallottino in 1984 named "Two-Q" (refer to Pallottino, 1984, Zhan, 1997 and Cherkassky, Goldberg and Radzik, 1993) on top of the prepared data structure. Please understand Pallottino's idea first, then implement the `Two-Q` algorithm. You may not use any third-party code or libraries for the routing function. Use of the standard libraries is allowed and encouraged. For utilities like path reconstruction, nearest node searching, however, it's okay to reuse existing third-party codes. In short, please focus more on the algorithm's implementation without being distracted by other aspects.

The routing function's interface will look like this:

```
function two_q(graph, source, target) -> route
```

where

- `graph` is the data structure containing nodes and links from the raw OSM network
- `source` is the origin node id of the path
- `target` is the destination node id of the path
- `route` is the found shortest route containing the nodes sequentially on the path

🍪 Optional dessert: visualization

If you still have time and energy, it would be great to visualize the paths found with the `Two-Q` algorithm on a digital map in an elegant way. This is a bonus task. And as for the "appetizer", please feel free to make use of open-source third-party libraries.

Other remarks

Please:

- send us your solution and the code packaged into a single .zip archive **within 7 days**
- feel free to choose your technical stack and programming language
- include a README document in your codebase to explain your technical decisions
- beware of version controlling of your codes and documents
- don't publish this challenge or your solution onto any code repository platform publicly
- don't include any commercial software library, service or component as dependencies

References

[1]: Pallottino, S. (1984) Shortest-Path Methods: Complexity, Interrelations and New Propositions. Networks, 14,257-267.
[2]: Zhan, F. B. (1997) Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures. Journal of Geographic Information and Decision Analysis, vol.1, no.1, pp. 69-82
[3]: Cherkassky, B. V., Goldberg, A. V., and Radzik, T. (1993) Shortest Paths Algorithms: Theory andExperimental Evaluation. Technical Report 93-1480, Computer Science Department, Stanford University.