BFST F2022 Examination Project Description

BSWU, IT University of Copenhagen
Troels Bjerre Lund

March 1, 2022

1 Introduction

This is the ITU BFST-F2022 course's examination project description. Based on this description, you must produce (a) an runnable program including all source code (preferably as an executable .jar file with everything included) as well as a full copy of the git repository and (b) a project report. You must submit the program and source electronically via learnit no later than 2 PM on Friday the 13th of May 2022 through ITU's learning platform learnit. You must hand the report in electronically via learnit a week later, no later than 2 PM on Friday the 20th of May 2022. Both source code and project report must be the product of groupwork as indicated on the course homepage. Submitting individually is not possible.

In this project, you must design and implement a system for visualizing and working with map data in the OpenStreetMap .OSM format.

2 Requirements for the final product

The final system must fulfill the following requirements:

1. allow the user to specify a data file for the application. The system should support loading a zipped .OSM file, but you may support other formats as well.
2. include a default binary file embedded as a resource of the program, which is loaded if the user does not choose another input file at start-up.
3. draw all roads in the loaded map dataset, using different colors for the different types of roads in the dataset.
4. show any rectangle of the map, as indicated by the user.
5. show the current zoom level, e.g., in the form of a graphical scale bar.
6. allow the user to change the visual appearance of the map, e.g., toggle color blind mode, or customize which elements to show.
7. where appropriate, draw other cartographic elements.
8. adjust the layout when the size of the window changes.

9. feature a coherent user interface for your map and accompanying functionality. Specifically, you must decide how the user can interact in a user-friendly and consistent manner with the user interface.
10. unobtrusively show (either continuously or on hover) the name of the road closest to the mouse pointer, e.g., in a status bar.
11. allow the user to search for addresses typed as a single string, and show the results on the map. The program must handle ambiguous user input appropriately, e.g., by displaying a list of possible matches.
12. be able to compute a shortest route on the map between two points somehow specified by the user (e.g., by typing addresses or clicking on the map).
13. allow the user to choose between routes for biking/walking, and cars. Car routes should take into account speed limits/expected average speeds. A route for biking/walking should be the shortest and cannot use highways.
14. output a textual description of the route found in Item 1 giving appropriate turn instructions. E.g., "Follow Rued Langgaardsvej for 50m, then turn right onto Røde Mellemvej." It should be possible to copy the description (as plain text) to the clipboard.
15. allow the user to add something to the map, e.g., points of interest.
16. be fast enough for convenient use, even when working with a complete data set for Denmark. The load time for .OSM can be hard to improve, but loading a binary file should be fast. After the start-up the user interface should respond reasonably smooth. (The lectures on spatial data structures and on implementing Dijkstra will be useful to achieve this.)

You may reuse any code written by a group member in the individual hand-ins in the final product, though it might be better to redesign the system from scratch. Project diaries by the group can also form the basis for the final report.

3 Extensions

Here is a list of possible extensions that you may consider doing for extra credit. Feel free to add some of these, or other features that you would like your application to support, but make sure that the basic functionality works first.

- Implement approximate matching for address searches, so "Ruud Langårds vej" matches to "Rued Langgaards Vej". (We do not expect this to be super fast for large data sets: Whether such searches can have sub-linear time complexity is an open research question.)
- Implement a "search-as-you-type" feature when searching for a road name. This would provide the user with a drop-down list of road names that match the characters of the road name he has typed so far.
- Clickable route description, which would update the view to show the selected turn on the map. It might be relevant to either show direction of the turn, or rotate the map to show the direction of the turn.

- Allow the user to drag and drop start and end point of the route, with instantaneous update of the shortest path, i.e., without rerunning Dijkstra.
- Allow the user to specify intermediary destinations, either by address search or by clicking on the map. Bonus points for supporting dragging of the current route.
- Make your route planner obey restrictions on turns (e.g. no left turn).
- Convert the current view to a PDF file.
- Explore methods in the literature to make the route planning more efficient. (Google Maps and similar services use such methods to quickly serve requests.) You can find relevant information in:
  `http://link.springer.com/content/pdf/10.1007/978-3-642-02094-0_7` Contraction Hierarchies can give enormous improvements on computation speed, for a reasonable amount of work.
- Use the Twitter API (`https://dev.twitter.com/docs/api/1/get/search`) to retrieve and display geotagged tweets on the map.
- ...

4 Rules for the hand-ins

- You are free to use third party libraries (apart from map APIs) as long as you cite your sources and reflect on your choice. Ask the lecturers if in doubt. You must include used libraries in your .jar file, if you choose to handin a .jar file.
- All members of the group are accountable for the entire code base at the exam. All members should be intimately familiar with all pieces of the basic system, as well as how the pieces work together. Individual group members may program some extensions alone, or in subgroups. The other group members must review any such code, preferably under the guidance of the authors.
- The final code submission must be in the form of a gradle project, where the `run` task starts the program, or as an executable jar-file with all needed libraries embedded (not counting JavaFX). The .jar file must be runnable without any arguments, in which case the program must load the default map from embedded binary files. You should pick something small enough that it loads fast, but is big enough to showcase your program.
- You must hand in a zip file of your repository. Remember to include your unit testing code.
- The final report submission must be a single PDF file, and must be prepared using LaTeX.
- The project report should describe both process and product (including parts that you handed in). It must be no longer than 40 pdf pages, excluding appendices.

4.1 Evaluation of project work

An important part of the evaluation will concern the report's **description of product and process**, cf. the course's intended learning outcomes. You must document your work in diary and a thorough git log, that you work in a structured manner, analyze the problems and make decisions, make plans and try to follow them, and document experiments. Your use of the local git-repository is documentation of your process, so make extensive use of it, including the wiki associated with your group repository.

4.2 Form of the report

The project report must be in English and should contain at least the following.

- Front page with project title, group number, names and emails of group members, hand-in date, name of course ("First-Year Project, Bachelor in Software Development, IT Univ. of Copenhagen").
- Preface (where, when, why).
- Background and problem area, data set, your requirements for the product.
- Analysis, arguments, and decisions about the design of the user interface.
- Analysis, arguments, and decisions about choices of algorithms and data structures. Also include descriptions of any experiments you have made to decide on choices on algorithms and data structures. (Such descriptions can benefit from running time and memory consumption comparisons.)
- Short technical description of the structure of the program, e.g., by using simple UML diagrams.
- Systematic test and benchmark of the resulting system. The test must be unit tests of classes and methods that you have implemented. The coverage must be extractable using the JaCoCo tasks in gradle, and you should discuss these briefly in the report. Benchmarks should include running times for common tasks (redrawing, route finding, loading) and give an overview of memory consumption of different object types in typical use cases (this data can be collected, e.g., using VisualVM).
- User manual for the system (brief, e.g., using cropped screen shots complemented by a description of functionality).
- Product conclusion: the extent to which the product fulfills your requirements, including a list of what is missing, and a brief description of which new ideas to functionality, design, and implementation that you have but have not explored.
- Process description with reflection on the process (including a description of how you could have improved the process).

The appendices should contain:

- Group norms (constitution).

- Diary / log book, including minutes of meetings.
- The change-logs for the releases.
- Optionally, selected source code for central parts that you refer to in the technical description.

5 Advice

- Remember that when you are writing the report, you have been working for a long time on this project; you might forget how little outsiders (such as the external examiner) understand about the data set, the graph representation, the problems of visualization, etc. You can signifcantly improve the quality of your report by using drawings and diagrams where appropriate.
- It can be a challenge to fit the report within the page limit; you should not do so by changing margins, using small fonts, etc., but rather by cutting away the least relevant parts. If in doubt, ask the lecturers.
- On the other hand, if you can deliver all relevant information in less than 40 pages, there is no need to pad the report to 40 pages. On the contrary, redundancy will make your report worse. In general, be concise and precise, focus on the parts that are specific to your project and choices, and avoid general statements (e.g., "most users appreciate good interface design").
- It is crucial to structure the text using sections, subsections, paragraphs, tables, figures, list of references, etc. It may help to define the document structure before starting to write. Please follow the listed order of the sections as much as possible.
- When you discuss a problem, remember to include pros and cons for different possible solutions, and an argument for your choice. You should limit this to non-trivial choices; e.g., do not bother explaining why you did not keep the entire map in a simple linked list.
- Make sure that you have implemented the core requirements before implementing extensions.
- Use a step-wise refinement procedure for your product. First analyze, design and implement a simple solution, then move to a more complicated solution.
- Remember to use what you have learned in BADS when arguing for choice of algorithms and data structures! Also remember that empirical observations about the data set is useful to argue for or against different possible solutions. Likewise, timing experiments are useful in arguing for one solution possibility over another.
- Make sure that everyone in the group has ownership of the code, i.e., understands what is going on and is confident to make changes. One way to ensure this is to explain parts of the program to each other (this is also a good way to test if the code is understandable).
- Make sure that each group member writes, and commits to the repository, an equal part of the final code.
- Have fun!