BFST F2017 Examination Project Description

BSWU, IT University of Copenhagen
Troels Bjerre Lund

27. februar 2017

1 Introduction

This is the ITU BFST-F2017 course's examination project description. Based on this description, you must produce (a) an executable .jar file which also contains all source code and (b) a project report. The program and source must be submitted electronically via learnit no later than 2pm on Tuesday the 16th of May 2017 through ITU's learning platform learnit. The report must be handed in electronically via learnit a week later, no later than 2pm on Tuesday the 23rd of May 2017. Both source code and project report must be produced by groups as indicated on the course homepage. It is not possible to submit individually.

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 should fulfill the following requirements. The system must:

1. allow the user to specify a data file to be used for the application. As a minimum, the system should support loading a zipped .OSM file.
2. allow the user to save and load the current model to and from a binary format which must be documented in the report.
3. allow the user to add something to the map, e.g., points of interest.
4. include a default binary file embedded in the jar file, to be loaded if no other input file is given at start-up.
5. draw all roads in the loaded map dataset, using different colors to indicate types of road.
6. where appropriate, draw additional cartographic elements.
7. allow the user to focus on a specific area on the map, either by drawing a rectangle and zooming to it, or by zooming in the map and pan the map afterwards.
8. adjust the layout when the size of the window changes.

9. unobtrusively show (either continuously or on hover) the name of the road closest to the mouse pointer, e.g., in a status bar.
10. allow the user to search for addresses typed as a single string, with the results being shown on the map. Ambiguous user input must be handled appropriately, e.g., by displaying a list of possible matches.
11. be able to compute a shortest path on the map between two points somehow specified by the user (e.g., by typing addresses or clicking on the map).
12. allow the user to choose between routes for biking/walking and for 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.
13. feature a coherent user interface for for your map and accompanying functionality. Specifically, you must decide how the mouse and keyboard is used to interact in a user-friendly and consistent manner with the user interface.
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."
15. indicate the current zoom level, e.g., in the form of a graphical scalebar.
16. allow the user to change the visual appearance of the map, e.g. , toggle color blind mode.
17. be fast enough that it is convenient to use, even when working with a complete data set for all of Denmark. The load time for .OSM can be hard to improve, but after the start-up the user interface should respond reasonably quickly.

Any code that has been written by a group member in the first hand-ins can be reused 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 additional 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" is matched to "Rued Langgaards Vej". While reasonably easy to do for small data sets, it is an open research problem how to do this efficiently in general.
- 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 an 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).
- Print current view to PostScript file (or directly to printer).
- Use the Twitter API (`https://dev.twitter.com/docs/api/1/get/search`) to retrieve and display geotagged tweets on the map.
- Explore methods in the literature to make the route planning more efficient. (Such methods are used by Google Maps, and similar services, to quickly serve requests.) You can find relevant information in:
  `http://link.springer.com/content/pdf/10.1007/978-3-642-02094-0_7`

As mentioned in Section 1, an important part of the final part of the project work consists of the group discussing which functionality you will implement and in which order and that the report describes your analysis (including arguments for your choices) and the extent to which you have succeeded in implementing the choices you have made. In other words, you should make it clear that you are working towards some goal (even if you don't make it) instead of randomly working on various possibilities.

4 Rules

- 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 lecturer if in doubt. Any used libraries must be embedded in your .jar file.
- All members of the group will be held 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. Extensions may be programmed by individual group members alone, or in subgroups. However, any such code must be reviewed extensively by the other group members, preferably under the guidance for the authors.
- The final code submission must be in the form of a single executable jar-file, with the source code and any needed libraries embedded. Remember to include your white and black box testing code. The .jar file must be runnable without any arguments, in which case a default map must be loaded from embedded binary files. You should pick something small enough that it loads fast, but is big enough to showcase your program.
- 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 previously handed in). It must be no longer than 40 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. Thus it is important that you 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 will be used as 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 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 regarding the design of the user interface.
- Analysis, arguments, and decisions regarding choices of algorithms and data structures. Also include descriptions of any experiments you have made to decide on choices on algorithms and data structures.
- Short technical description of the structure of the program using simple UML diagrams.
- Systematic test of the resulting system. The test can be concerned with unit tests of classes that you have implemented. Moreover, you should also include a white-box test of a part of your program that has non-trivial control flow (e.g., a method with nested `if` statements and `while`/`for` loops.)
- User manual for the system (brief, e.g., using 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 quite a long time on this project; hence it is easy to forget how little outsiders (such as the external examiner) understand about the data set, the graph representation, the problems regarding visualization, etc. Your explanation should be brief and to the point; try to use drawings and diagrams where appropriate.
- It can be a challenge to fit the report within the maximum number of pages; you should not do so by changing margins, using small fonts, etc., but rather by repeatedly cutting away the least relevant parts. If in doubt, ask the lecturer.
- Remember to structure the text using sections, subsections, paragraphs, tables, figures, list of references, etc.
- When you discuss a problem, remember to include pros and cons for different possible solutions, and an argument for your choice. However, you should limit this to non-trivial choices; e.g., don't bother explaining why you didn't just 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 can be used to argue for or against different possible solutions. Likewise, timing experiments can also be used to argue for one solution possibility over another.
- Make sure that everyone in the group has ownership of the program, 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).
- Have fun!