BFST F2014 Examination Project Description

BSWU, IT University of Copenhagen
Søren Debois & Troels Bjerre

April 3, 2014

1 Introduction

This is the text of the ITU BFST-F2014 course's examination project description. Based on this description, you must produce (a) source code and (b) a project report and submit both electronically via learnit no later than 2pm, May 21, 2014. 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 visualising and working with map data.

2 Requirements for the final product

The final system should fulfill the following requirements, which you presumably already designed and implemented.

- Draw all roads in the map dataset;
- use different colors to indicate types of road;
- adjust the drawing when the size of the window changes;
- focus on a specific area on the map, either by drawing a rectangle and zooming in it, or by zooming in the center of the map and pan the map afterwards
- unobtrusively show either continuously or on hover the name of the road closest to the mouse pointer, e.g., in a status bar.

Moreover, your product must also:

1. 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).
2. 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;

3. be fast enough that it is convenient to use, even when working with a complete data set for all of Denmark. The start-up time can be hard to improve as long as the data is stored in text files, but after the start-up the user interface should react reasonably quickly.

4. allow the user to choose between using either the KRAK dataset and Open Street Map's map of Denmark as found on, e.g., `http://download.geofabrik.de/europe/denmark.html`. The above requirement that the program is fast enough applies also to OpenStreetMap data.

5. contain at least one extension feature, either from the list in Section 3 below, or one you invent yourselves. In the latter case, contact a lecturer to check that your feature is substantial enough.

6. for groups of 4, your solution must also be able to 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."

Some of what you have programmed previously can be reused without change in the final product, but there are probably also parts that require rethinking and reworking. Likewise, a lot of what you have written in your work sheets and hand-ins can probably be used in the final report, but in a modified and rewritten form.

3 Extensions

As mentioned, you must implement at least one of these. Extra credit for doing more than one.

- (a) Make it possible to compute a shortest path tree from a given starting point p, and highlight the path in the tree from p to a point on the map defined by *double-clicking* on it. (This should be essentially instantaneuous once the tree is computed.) (b) You must also continuously the displayed shortest path to wherever you drag the mouse.

- Improve the looks of the outermost zoom levels. For example, compute an approximation of the coastline so you can shade water and land in different colours. Coastline data can be obtained from: `http://www.ngdc.noaa.gov/mgg_coastline/` or e.g. from: `http://download.geofabrik.de/europe/denmark.html`. Observer that both datasets are in Lat/Lon format.

- Use the Twitter API (`https://dev.twitter.com/docs/api/1/get/search`) to retrieve and display geotagged tweets on the map.

- Implement approximate matching for address searches, so "Ruud Langrds 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.

- Improve the user interface so that the user can 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.

- Make your route planner obey restrictions on turns (e.g. illegal left turn); this requires understanding (and parsing) of the file `turn.txt`, which is included in the Krak data set. (Hint: Construct a new input graph where an intersection with k adjacent road segments is represented by k nodes.)

- 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%2F978-3-642-02094-0_7`

You are encouraged to find additional or alternative extensions you find (more) interesting. Clear your custom extension with a lecturer, though.

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 a lecturer if in doubt.

- Extensions to the basic system may be programmed by individual group members alone, or in subgroups. Together with the oral exam, this can give a basis for individual grading. However, it is the responsibility of the whole group that the final product is a well-integrated whole, and that the software for the basic functionality is of good quality.

- The final submission must be in the form of a zip-file containing a PDF project report, source code, a README file indicating how to build and run your program. Contact a lecturer before submission if you application is not written in Java.

- 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. The report may contain parts describing an individual's contribution, but these parts should not exceed 4 pages per group member.

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 work sheets, that you work in a structured manner, analyze the problems and make decisions, make plans and try to follow them, and document experiments.

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 on how to implement the user interface, including 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). This must include an overview of who in the group (if not all members) were main contributors of different parts of the product.

The appendices should contain:

- Group norms (constitution).
- Diary / log book.
- Work sheets for parts of your project.
- Optionally, selected source code for central parts that you refer to in the technical description.

5 Advice

- Remember that you have already worked 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. Therefore, you should remember to explain these items in the report instead of just talking about "UTM32", "KrakNode", 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 a lecturer.
- Remember to structure the text using sections, subsections, paragraphs, tables, figures, list of references, etc.
- When you discuss a problem then remember to include pros and cons for different possible solutions, and an argument for your choice.
- 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!