DM823 String Algorithms

Fall 2013

Department of Mathematics and Computer Science
University of Southern Denmark

November 21, 2013

Introduction

The purpose of this project is try out in practice some of the algorithms learnt in this course.[1] The project is to be done in groups of two persons. Single person hand-ins are also allowed.

Requirements

You are to implement the following algorithms for exact pattern matching:

- The naive algorithm (page 29).
- The naive algorithm with the *last-occ* (page 31) heuristic added (this is called Horspool's algorithm).
- The KMP algorithm (notes).
- The BM algorithm (page 107).
- The BM algorithm with the *last-occ* heuristic added.
- The Galil variant of the BM algorithm (page 112).
- The Galil variant of the BM algorithm, with the *last-occ* heuristic added (for unsuccessful searches).

You are welcome to add further algorithms to this set, such as the algorithms of sections 3.5 and 3.6 in the book (using the *good-suff* table), or a version (to be developed by you) of BM that extends the Galil idea to also not repeating test for characters of the pattern x which after the shift resulting from an *unsuccessful* attempt is known to match (via the definition of *good-suff*).

Any necessary preprocessing of the pattern (the table of borders for KMP, *good-suff* for BM versions) is of course part of the implementation required.

[1] Alternatively, you can do a theoretical variant of the project, see later.

The choice of programming language is left to you.

After implementation, empirical investigations of the implementations should be performed on non-trivial size test data with varying alphabet sizes and data origins. These test data, as well as further details of the experiments required (suggested query patterns, number of experiments, etc.) will be given later by the lecturer.

Finally, the outcome of the experiments should be displayed appropriately, e.g. by graphs on running time (wall-clock time) versus pattern length for each data instance. The results should be discussed.

Formalities

You should hand in a report of 10–15 pages (including figures) in pdf-format, and your files with source code (source code is not to be included in the report, except possibly as snippets in the main report text). State the names of all group members on the front page of the report.

The main focus of the report should be on describing the design choices made during development and the structure of the final implementations, as well as on describing in detail the experiments made and discussing the outcome of these.

All code should be your own. In particular, grabbing code from the net is not legal. Basing you implementation on *pseudo-code* from the net (or elsewhere) is allowed, if you reference the original, and if you *explain* the working of the pseudo-code (if the pseudo-code is different from that/those in the course material). But the pseudo-code available in the book is very precise already.

The project will be evaluated by pass/fail grading. The grading will be based on:

- The clarity of the writing and of the structure of the report.
- The thoroughness of the experiments (execution as well as discussion).
- The amount of work done.

The material should be handed in using "SDU Assignment" in the course page in the Blackboard system. Submit your files (report and program source files) as one zip-archive.

You must hand in the material by

Friday, January 3, 2014, at 12:00

Variant

If you prefer, you may instead of the above do a theoretical project (no programming), in groups of size at most two. The report will then consist of a written exposition of a research paper relevant to the course. The length is expected to be 10–20 pages. You are to describe, in your own words (avoid as much as possible just copying from the paper), the background (briefly), the main result achieved, the algorithm of the result, and the proof of correctness and time complexity.

Some examples of possible papers are:

- M.I. Abouelhoda, S. Kurtz, E. Elbas. *Replacing suffix trees with enhanced suffix arrays.* Journal of Discrete Algorithms 2 (2004), p. 53–86. [Only sections 1–6 need to be covered. Not all applications discussed in paper need be covered.]
- S. J. Puglisi, W.F. Smyth, M. Yusufu. *Fast, Practical Algorithms for Computing All the Repeats in a String.* Mathematics in Computer Science 3 (2010), p. 373–389. [Only sections 1 and 2 need to be covered.]
- S. Dori, G. M. Landau. *Construction of Aho Corasick automaton in linear time for integer alphabets.* Information Processing Letters 98 (2006), p. 66–72.

You are free to suggest further papers (but they must be approved by the lecturer).