DM823 String Algorithms

Fall 2016

Department of Mathematics and Computer Science
University of Southern Denmark

October 5, 2016

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 one or two persons.

Algorithms

You are to implement the following algorithms for exact pattern matching (page numbers refer to material handed out along with the project description):

- 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).

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

The programming language should be one from the "C family" (Java, C, C++, C#, . . .).

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

Data Sets

A collection of 16 texts of various types and alphabet sizes has been created, and can be found in the file `ProjectDataSets.zip`. Each text is a single file, and the contents of the files are described in the file `INDEX.TXT`.

Experiments

You are to perform the following for each of at least 10 of the texts (of your own choice).

For each text, the pattern to search for should be m characters starting at a random position in the latter half of text. All occurrences should be found (not just the first). The values of m used should be: 4, 8, 12, 40, and $n/16$, where n is the text length.

For each combination of

- Algorithm implemented
- Pattern length m
- Text chosen

do the following:

1. Load the text into an array of bytes.
2. Select the pattern in the text (remember to align to a multiple of the bytes-per-character value) and copy it from the text into another array.
3. Read the time.
4. Preprocess the pattern (if the algorithm uses preprocessing).
5. Do the search for all occurrences.
6. Read time again and calculate elapsed time.

Repeat from point two enough times for the sum of the elapsed times to be around a second or more. However, at least five repetitions should always be done. Then take the average elapsed time (i.e., sum of elapsed times divided by number of repetitions).

During experiments, you should just count the number of occurrences (and report a single number in the end), in order not to spend processing time on output. During development, you for debugging reasons may also want to report the position of occurences (and test against the brute force method). Compile with maximal optimization flags (if available—Java has none).

Presentation of Results

You should display your results using grouped/clustered bar charts (see e.g. example two in example section of `http://www.burningcutlery.com/derek/bargraph/` for a definition). You should make one chart per text. The y-axis should be running time (average over all repetitions of search, cf. above), and the x-axis should contain groups of experiments, with the pattern length being the same within a group, and the members of a group being the different algorithms implemented.

To get a reasonable resolution in the diagrams: if the brute force is more than three times slower than the second slowest method, leave it out of the chart (but report in writing the factor it is slower than the second slowest).

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 non-trivial design choices (if any) made during development, and on describing the experiments made and in particular 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 handed out 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

Monday, December 19, 2016, 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).