Programming Project
Streaming Algorithms for XPath

Web Data Models

Silviu Maniu

October 17th, 2016

The objective of this programming assignment is to be able to implement and analyze efficient algorithms for fragments of XPath queries. For this, we will implement a subset of the algorithms presented in [GGM+04].

1 Preliminaries

This assignment will use the streaming format for the XML. An XML document will be given as input to the implemented algorithms a whitespace delimited file having the following format:

```
0|1 <whitespace> element1
0|1 <whitespace> element2
...
```

where `0|1` is a bit indicating a startElement or endElement event respectively, and `element` is the name of the element. For instance, the following file `example.txt` contains the document `<a><b></b></a>`:

```
0 a
0 b
1 b
1 a
```

The parameters of the implemented executables will be the following:

```
./<program_name> <xml_file> <xpath_query>
```

The console output will be the *preorder* ID (zero-indexed) of the matching nodes, one per line, or will be empty if no nodes match. For instance:

```
./<program_name> example.txt //a/b
```

will return the following output:

```
1
```

2 Assignments

2.1 Algorithm Implementation

Assignment 1 Implement a streaming algorithm for XPath queries of the form

$$//e_1/e_2/\cdots/e_n,$$

where e_i are element names.

Assignment 2 Implement a streaming algorithm using the *lazy* DFA method explained in [GGM$^+$04], for queries of the form:

$$//p_1//p_2//\cdots//p_n,$$

where each p_i is an path of the form:

$$e_{i1}/e_{i2}/\cdots/e_{im},$$

and e_{ij} are element names.

An important note for both implementations: the document present in the input file *should not be loaded in memory*. Both algorithms can be implemented and the queries can be matched in one pass over the file. On the other hand, you are allowed to keep additional data structures and the query in memory.

2.2 Analysis & Experimental Evaluation

For the two algorithms implemented in the above assignment, write a document containing an implementation analysis and an experimental section.

The *implementation* analysis section will contain a write-up of the implementation choices for each of the algorithms. For example, it should contain an analysis of the data structures used, but also a discussion of the optimizations used in the implementation.

The *experimental evaluation* analysis will contain an empirical evaluation of the algorithm in the form of plots evaluating the execution time and memory space for a selection of document and query sizes. For instance, a possible plot has the size of the document on the x-axis and the execution time on the y-axis. The above are just suggestions: other evaluations are welcome and appreciated – if they are relevant to the implementations.

3 Submission & Evaluation

Send your submissions by email to Silviu Maniu (`silviu.maniu@lri.fr`) by **Friday, November 18th 2016, 11:59pm**, for full credit. Submissions sent by Saturday, November 19th 2016, 11:59pm will incur 5 points of penalty out of 20. Submissions received after this date will receive no credit.

Your submission should contain two files:

1. an archive (.zip, .tar, .rar, etc.) consisting of the source code and instructions on how to compile (if necessary) and execute the code, and

2. a PDF document containing the analysis and experiments document.

Your submission will be evaluated based on private tests run on your algorithms for correctness, on the evaluation of the code and implementation quality, and the quality of your evaluation document. The implementation and documentation will have equal weight in the final grade.

References

[GGM$^+$04] Todd J Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM TODS*, 29(4):752–788, 2004.