Fast Construction of a Word↔Number Index for Large Data

Miloš Jakubíček, Pavel Rychlý, Pavel Šmerk

Natural Language Processing Centre
Faculty of Informatics
Masaryk University

7. 12. 2013

Introduction

- Inspiration: Aleš Horák @ 1st NLP Centre seminar :-)
  - (but we still did not compare Manatee and some sql DB)
- Problem: indexes for large text corpora (billions of tokens)
- Current solution: `.lex`, `.lex.idx` and `.lex.srt` files
  - `.lex`: null-terminated strings, in the order of appearance in corpus
  - `.lex.idx`: 4B offsets of words in `.lex`
  - `.lex.srt`: 4B indices (positions in `.lex.idx`) sorted alphabetically
  - id2str: 2 accesses to the memory
  - str2id: 3 * $\ln_2$ |lexicon| accesses to the memory
- New solution: HAT-trie + (reimplemented) Daciuk's fsa tools
  - HAT-trie: cache-conscious, combines trie + hash, allows sorted access
    - for indexing natural language strings, it is among the best solutions regarding both time and space
  - Daciuk: minimal DAFSA for perfect hashing

Data sets used in the experiments

data set	size	words	unique	size	language
100M	1148 MB	110 M	1660 k	31 MB	Tajik
1000M	5161 MB	957 M	1366 k	14 MB	French
10000M	69010 MB	12967 M	27892 k	384 MB	English

- three sets of corpus data: they differ not only in size
  - Tajik uses Cyrillic $\Rightarrow$ words are two times longer only due to encoding
  - French corpus (OPUS project): mostly legal texts $\Rightarrow$ limited vocabulary

Comparison of encodevert and hat-trie

	encodevert		hat-trie		
data set	**time**	**memory**	**time**	**memory**	**size**
100M	3:11 m	0.44 GB	26.5 s	0.06 GB	44 MB
1000M	23:01 m	0.40 GB	2:21 m	0.04 GB	25 MB
10000M	7:38 h	0.98 GB	44:37 m	0.78 GB	607 MB

	encodevert		hat-trie
data set	**local**	**fair**	**fair**
100M	3:27 m	1:25 m	32.6 s
1000M	26:10 m	6:26 m	3:09 m
10000M	9:21 h	4:02 h	1:02 h

- the table from the paper have revealed to be unfair to encodevert
- **local** data on local hdd, but probably more used
- **fair** times: both apps produces the same set of files
  - in fact, this is still unfair, but now to hat-trie
  - $\Rightarrow$ whole applications have to be tested

Reduction of the size of data

	`encodevert`		**hat-trie**		
data set	**time**	**memory**	**time**	**memory**	**size**
100M	3:11 m	0.44 GB	26.5 s	0.06 GB	44 MB
1000M	23:01 m	0.40 GB	2:21 m	0.04 GB	25 MB
10000M	7:38 h	0.98 GB	44:37 m	0.78 GB	607 MB

	`fsa_ubuild`		**hat + new fsa**		
data set	**time**	**memory**	**time**	**memory**	**size**
100M	*failed*		31.7 s	0.09 GB	15 MB
1000M	15:48 m	0.11 GB	2:34 m	0.06 GB	11 MB
10000M	7:44 h	31.01 GB	1:08 h	1.47 GB	363 MB

- for very large corpora the files can consume a lot of memory
- with Daciuk's fsa tools we have built automata for perfect hashing
  - `fsa_ubuild` is an original Daciuk's implementation (unsorted data)
  - **hat + new fsa** is an reimplementation with HAT-trie as presort
- (experiments from the two tables were run on different hardware)

HAT-trie based sort + fsa overperforms `fsa_ubuild`

	`fsa_ubuild`		hat + new fsa		
data set	**time**	**memory**	**time**	**memory**	**size**
100M	*failed*		31.7 s	0.09 GB	15 MB
1000M	15:48 m	0.11 GB	2:34 m	0.06 GB	11 MB
10000M	7:44 h	31.01 GB	1:08 h	1.47 GB	363 MB

	hat-trie sort		`fsa_build`		**new fsa**	
data set	**time**	**memory**	**time**	**memory**	**time**	**memory**
100M	28.4 s	0.06 GB	12.4 s	0.21 GB	4.2 s	0.03 GB
1000M	2:51 m	0.04 GB	5.6 s	0.11 GB	1.8 s	0.03 GB
10000M	59:16 m	0.77 GB	35:15 m	27.07 GB	9:36 m	0.71 GB

- the second table compares fsa construction from sorted data
- $\Rightarrow$ having such an effective sort algorithm, to sort data and then use the algorithm for sorted data is always better than `fsa_ubuild`
- $\Rightarrow$ to reduce the used memory it would better to flush sorted data to hard disk before fsa construction, as the time penalty is minimal

Future Work

- it is a work in progress, even the measured times are biased
- we want to
  - fine tune hat-trie (we have used default settings)
  - further reduce
    - compile space: fsa can be built directly in memory
    - compile time: hash for "registered" nodes
    - run space: VLEncoded information, relative adresses, UTF-8, ...
    - run time: smaller run space, numbers in arcs
  - run experiments on a hdd not shared with other processes