11 – Indexing and Searching Very Large Texts

IA161 Advanced Techniques of Natural Language Processing

M. Jakubíček

NLP Centre, FI MU, Brno

November 27, 2017

1 Indexing

2 Searching

Searching big text corpora

Corpus:

- positional attributes – word form, lemma, PoS tag, . . .
- structures and structure attributes – documents (e.g. with author, id, year, . . .), paragraph, sentence
- searching: Manatee/Bonito/Sketch Engine
- `http://corpora.fi.muni.cz`
- `http://the.sketchengine.co.uk`

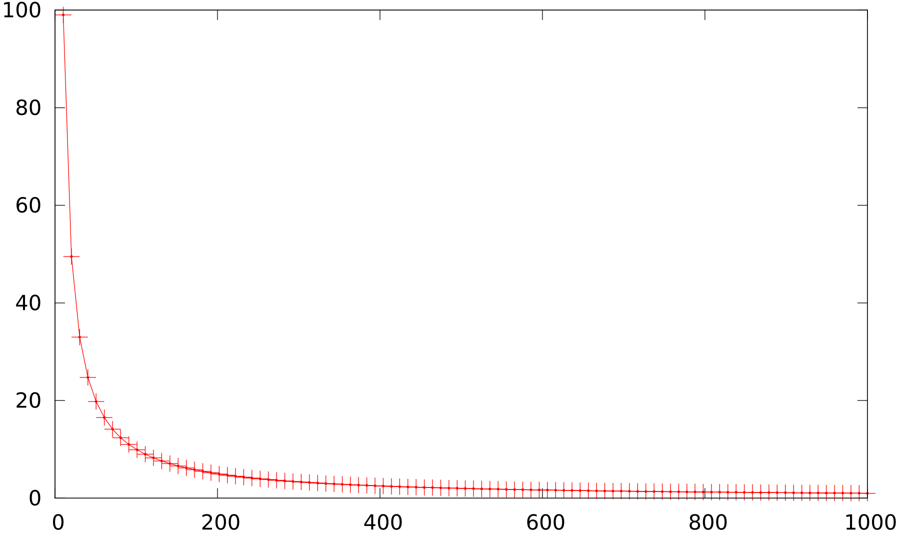
Searching big text corpora

- data too big to be stored in memory
- data too big to be search sequentially

⇒ preprocessing needed (indexing, alias corpus compilation)

- key decisions are:
  - trade off between compile-time (preprocessing) and run-time
  - trade off between in memory and off-memory processing

Zipf's law I



Zipf's law II

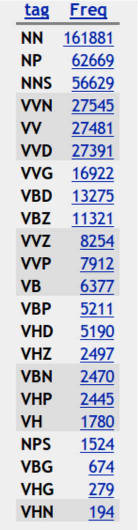
- may be simplified to inductive definition:

Zipf's law (simplified)

frequency of the n-th element $f_n \approx \frac{1}{n} \cdot f_1$

- ⇒ frequency is inversely proportional to the rank according to frequency
- ⇒ one needs really large corpora to capture all the variety of many language phenomena
- ⇒ implications for text indexing

Zipf's law III



tag	Freq
NN	161881
NP	62669
NNS	56629
VVN	27545
VV	27481
VVD	27391
VVG	16922
VBD	13275
VBZ	11321
VVZ	8254
VVP	7912
VB	6377
VBP	5211
VHD	5190
VHZ	2497
VBN	2470
VHP	2445
VH	1780
NPS	1524
VBG	674
VHG	279
VHN	194

Substantives + Verb tags on the Brown corpus

Building corpora

1. content definition (what will it be used for? how do I get texts?)
2. obtaining data (e.g. crawling)
3. data cleaning (spam, boilerplate, duplicates)
4. tokenization
5. sentence segmentation
6. further annotation (PoS tagging)
7. corpus indexing and analysis

Building corpora

1. content definition (what will it be used for? how do I get texts?)
2. obtaining data (e.g. crawling)
3. data cleaning (spam, boilerplate, duplicates)
4. tokenization
5. sentence segmentation
6. further annotation (PoS tagging)
7. corpus indexing and analysis

Corpus indexing

- text corpus is a database
- standard (=relational) database management systems are not suitable at all
    - text corpus does not have relational nature
- special database management systems needed

⇒ Manatee

Indexing corpora in Manatee

Key data structures for a positional attribute:

- lexicon
    - because operations on numbers are just so much faster than on strings
- corpus text
    - to iterate over positions
- inverted (reversed) index
    - to give fast access to positions for a given value

How to store integer numbers

- given Zipf's distribution: fixed-length storing very inefficient
- variable-length more complicated but yielding much smaller and quicker indices
- variable-length bit-wise universal Elias' codes: gamma, delta codes
- cf. Huffman coding

Indexing corpora in Manatee

Structures and operations:

- operations in between: string (`str`) – number (`id`) – position (`poss`)
- lexicon building: ⇒ word-to-id mapping ⇒ operations on numbers, not strings ⇒ `id2str, str2id`
- inverted index: `id2poss`
- corpus text: `pos2id`
- yields transitively also `pos2str, str2poss`

Searching corpora in Manatee

- key idea: operations on sorted forward-only streams of positions
- FastStream – single position stream
- RangeStream – stream of position pairs (structures: *from* position, *to* position)

CQL

- = Corpus Query Language (Christ and Schulze, 1994)
- positions and positional attributes: `[attr="value"]`
- structures and structural attributes: `<str attr="value">`
- example:

```
[word=".*ing" & tag="V.*"]
   <doc id="20[5-9].*"
```

- established a `within <str/>` query:

```
[tag="N.*"]+ within <s/>
```

  and alternative `meet/union` query:

```
(meet [lemma="take"] [tag="N.*"] -5 +5)
   (union (meet ...)  (meet ...))
```

CQL in Manatee/Bonito

- ehnancements and differences to the original CQL syntax
- `within <query>` and `containing <query>`
- `meet/union` (sub)query
- inequality comparisons
- frequency function

within/containing queries

- searching for particles:

  ```
  [tag="PR.*"] within [tag="V.*"] [tag="AT0"]?
  [tag="AJ0"]* [tag="(PR.?|N.*)"] [tag="PR.*"] within
  <s/>
  ```

- searching for a Czech idiom “hnout někomu žlučí” (“to get somebody’s goat”):
  word-by-word translated as:
  hnout “move” [V, infinitive]
  někomu “somebody” [N, dative]
  žlučí “bile” [N, instrumental].

  ```
  <s/> containing [lemma="hnout"] containing
  [tag=".*c3.*"] containing [word="žlučí"]
  ```

within/containing queries

- structure boundaries: begin: `<str>`, whole structure: `<str/>`, end: `</str>`
- changes: `within <str>` not allowed anymore, use `within <str/>`

meet/union queries

- combined with regular query: <s/>

  ```
  containing (meet [lemma="have"] [tag="P.*"] -5 5)
  containing (meet [tag="N.*"] [lemma="blue"])
  ```

- changes: meet/union queries can be used on any position, they can contain labels and no MU keyword is required (and deprecated):

  ```
  (meet 1:[] 2:[]) & 1.tag = 2.tag
  ```

Inequality comparisons

- former comparisons allowed only equality and its negation: `[attr="value"] [attr!="value"]`
- inequality comparisons implemented: `[attr<="value"] [attr>="value"] [attr!<="value"] [attr!>="value"]`
- intended usage: `[tag="AJ.*"] [tag="NN.*"] within <doc year>="2009">`
- sophisticated comparison performed on the attribute value: `<doc id<="CC20101031B">` matches e.g. `BB20101031B`, `CC20091031B`, `CC20101030B CC20101031A`.

Fixed string comparisons

- normally the CQL values are regular expressions
- sometimes this is not desirable (batch processing needs escaping of metacharacters)
- new == and !== operator introduced for fixed strings comparison
- no escaping needed except for ’”’ and ’\’
- examples: ”.”, ”$”, ” ” matches a single dot, dollar sign and tilda, respectively, ”\n” matches a backslash followed by the character n,

Frequency function

- a frequency constraint allowed in the global conditions part of CQL:
  `1:[tag="PP.*"] 2:[tag="NN.*"] & f(1.word) > 10`

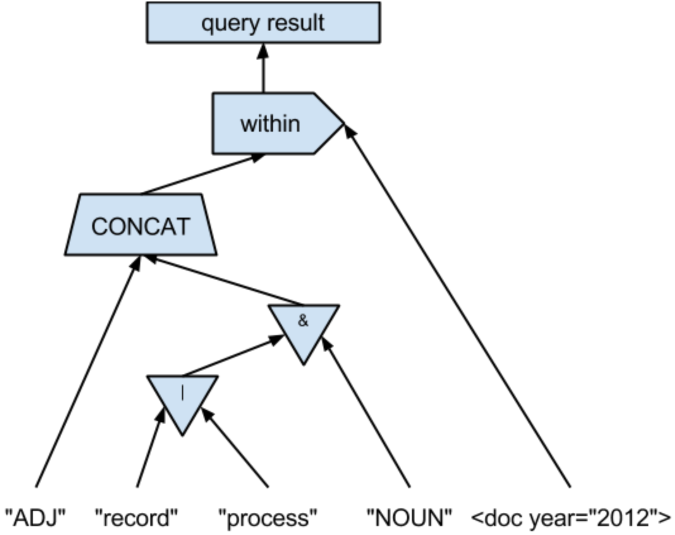
Performance evaluation

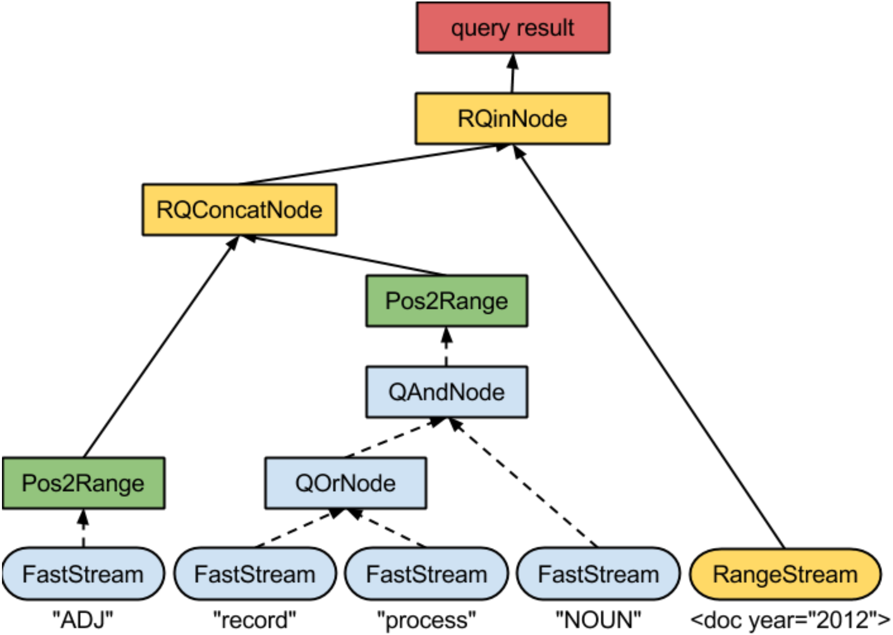
Table: Query performance evaluation – corpora legend: ○ BNC (110M tokens), • BiWeC (version with 9.5G tokens), ∗ Czes (1.2G tokens)

query	# of results	time (m:s)
○ `[lemma="time"]`	179,321	0.07
○ `[lemma="t.*"]`	14,660,881	3.12
○ Ex: particles	1,219,973	33.36
• Ex: particles	97,671,485	32:26.48
∗ Ex: idioms	66	1:6.86
○ Ex: meet/union	3	8.47
• Ex: meet/union	1457	7:13.12

CQL query evaluation

Example: `[tag="ADJ"] [(word="record" | word="process") & tag="NOUN"] within <doc year="2012"/>`




query result
RQinNode
RQConcatNode
Pos2Range
QAndNode
Pos2Range
QOrNode
FastStream
"ADJ"
FastStream
"record"
FastStream
"process"
FastStream
"NOUN"
RangeStream
<doc year="2012">

Today's Corpora in Sketch Engine

- **LARGE** (= billions of tokens, and it's going to be worse)
- complex multi-level multi-value annotation
- wide range of languages
- growing demand on complex searching – moving from morphology to syntax and semantics
- search API for automatic information retrieval and post-processing in particular applications needed