05 – Indexing and Searching Very Large Texts

IA161 Natural Language Processing in Practice

M. Jakubíček

NLP Centre, FI MU, Brno

November 21, 2023

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`
- `https://app.sketchengine.eu`
- SQL unsuitable (independent rows)

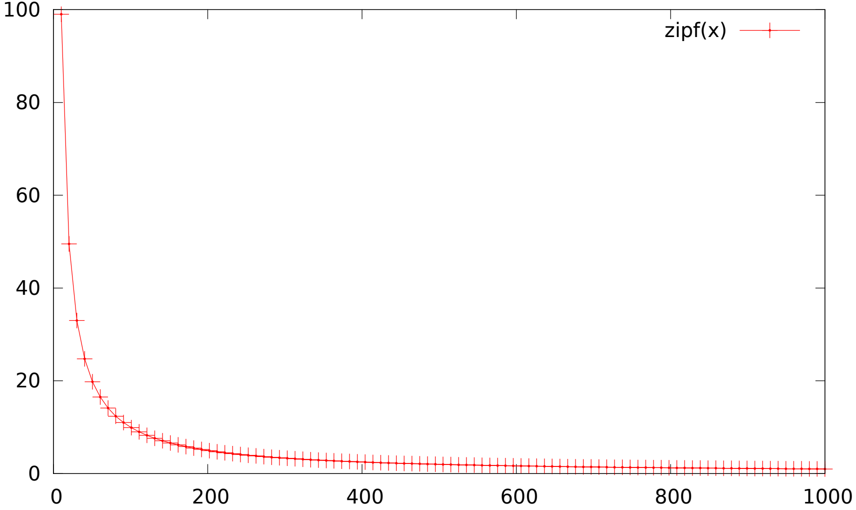
Searching big text corpora

- data too big to be stored in memory
- data too big to be searched 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

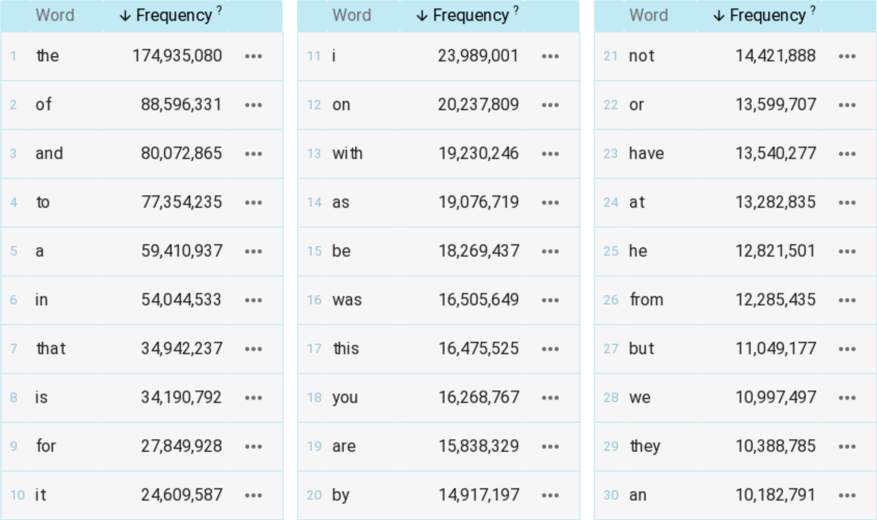
- may be simplified to inductive definition:

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

Zipf’s law III



	Word	↓ Frequency ?	
1	the	174,935,080	•••
2	of	88,596,331	•••
3	and	80,072,865	•••
4	to	77,354,235	•••
5	a	59,410,937	•••
6	in	54,044,533	•••
7	that	34,942,237	•••
8	is	34,190,792	•••
9	for	27,849,928	•••
10	it	24,609,587	•••

	Word	↓ Frequency ?	
11	i	23,989,001	•••
12	on	20,237,809	•••
13	with	19,230,246	•••
14	as	19,076,719	•••
15	be	18,269,437	•••
16	was	16,505,649	•••
17	this	16,475,525	•••
18	you	16,268,767	•••
19	are	15,838,329	•••
20	by	14,917,197	•••

	Word	↓ Frequency ?	
21	not	14,421,888	•••
22	or	13,599,707	•••
23	have	13,540,277	•••
24	at	13,282,835	•••
25	he	12,821,501	•••
26	from	12,285,435	•••
27	but	11,049,177	•••
28	we	10,997,497	•••
29	they	10,388,785	•••
30	an	10,182,791	•••

enTenTen2008, 3.2G tokens

Zipf's law IV

About 1 billion words is enough to have enough evidence for single word units. But not for multiwords:

word	**Brown (1M)**	**BNC (100M)**	**enTenTen08 (2.7G)**	**enTenTen15 (15.7G)**
carbonation	0	5	429	2,817
weird phrase	0	0	14	34

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

REGEX cheat sheet

.

exactly one unspecified character

`w.n`	win won wen wun wan
`ca.`	cat car cap cab can

*

zero or more occurrences of the preceding character

`a*h`	h ah aah aaah aaaah
`c.*ing`	words starting *c-* and ending *-ing*: *cooking*
`*ool`	error: missing character before asterisk
`c.*`	word starting *c-*

?

preceding character is optional

`colou?r`	color colour
`bet?`	be bet

^

not, brackets are compulsory

`[^m]et`	pet get bet let (but not *met*)
`[^mpg]et`	set let (but not *met pet get*)

( )

grouping, prioritizing

`meter|re`	*meter* or *re*
`met(er|re)`	*meter* or *metre*

[ ]

list, range – one character from content of the brackets

`[mpgb]et`	met pet get bet
`m[2-5]`	m2 m3 m4 m5
`[0-9]*`	any number of digits
`[a-z]`	one lowercase letter
`[A-Z]*`	any number of uppercase letters: UNESCO, UK, WIFI
`[A-Za-z]*`	any number of letters (but not numbers)

|

OR – the characters to the left or those on the right

`get|met`	*get* or *met*
`met(er|re)`	*met**er*** or *met**re***

+

one or more occurences of the preceding character (compare *)

`a+h`	ah aah aaah aaaah
`hallo+`	hallo halloo hallooo halloooo

{ }

repetition

`gr{2,4}`	*grr grrr grrrr*
`[A-Z]{3}`	3-letter acronyms
`(bla){2,3}`	blabla blablabla

\

escaping, removes or adds special meaning to a character

`...`	finds 3-letter words
`\.\.\.`	finds three dots
`\w`	finds any letter

REGEX character classes

These classes also cover non-English (Unicode) characters, e.g. ñ ř ç 香 ж ش

		shortcut	NOT
`[[:alpha:]]`	any letter including Unicode	`\w`	`\W`
`[[:digit:]]`	any digit, equivalent to [0-9]	`\d`	`\D`
`[[:alnum:]]`	any digit or letter including Unicode		
`[[:lower:]]`	any lowercase letter including Unicode		
`[[:upper:]]`	any uppercase letter including Unicode		
`[[:punct:]]`	punctuation -!"#$%&'()*+,./:;<=>?@]_`{[		
`[[:space:]]`	whitespace character space, new line, tab, carriage return	`\s`	`\S`

REGEX examples

`k.*`	words starting with *k*
`.*k.*`	starting, containing or ending with *k* (including just *k*)
`.+ment`	words ending with *-ment* (but not just *ment*)
`[^x]+x[^x]+`	words containing *x* but not starting or ending with it
`[[:upper:]][[:lower:]]*`	words starting with one capital letter
`[[:upper:]]*`	acronyms incl. unicode: *EU, SRPŠ, ЖЭК*
`[a-z]*\d[a-z]*`	lowercase words containing a digit: *face2face*
`(kilo|centi)?metre`	*kilometre centimetre metre*
`dog|.*cat|mouse`	*dog* OR *cat* OR *pussycat* OR *tomcat* etc. OR *mouse*

Extended regex manual on http://ske.li/regex

For complete information, google "regular expressions cheat sheet or tutorial"

SKETCH ENGINE

CQL cheat sheet

sample syntax

```
[lemma="go"]
[lemma="work" & tag !="V.*"]
[tag="N.*"][ ]{1,5}[tag="V.*"] within <s/>
```

& joins two or more conditions for the same token

{ } `[word="ha"]{3}` finds *ha ha ha*
`[tag="N.*"]{2,5}` finds 2, 3, 4 or 5 nouns

? makes the preceding token optional
`[lc="new"][lc="cheap"]?[lc="phone"]`
finds both *new phone* and *new cheap phone*

| `[lemma="accommodate"]|[lemma="put"][lc="up"]` finds *accommodate* **or** *put up*

( ) the tokens inside behave as one group
`[lc="might"]([lc="as"][lc="well"])? [tag="V.*"]`
finds both *might as well go* and *might go*

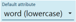
***** unlimited (max. 100) repetitions of the preceding token
`<s> [ ]* [word="\?"]</s> within <s/>` finds sentences finishing with a question mark

<> used for structures such as documents, paragraphs and sentences: `<s>` beginning ▪ `</s>` end ▪ `<s/>` all

~ searches for *chop* followed by *carrot* and its 15 most similar nouns (vegetables)
`[lemma="chop"] []{0,3} ~15"carrot-n"`

Default attribute

Makes queries easier to read. It is applied to each token without square brackets. This query
`[lc="might"]([lc="as"][lc="well"])? [tag="V.*"]`
can be simplified like this:



Default attribute
word (lowercase)

`"might"("as" "well")? [tag="V.*"]`

`within <s/>`

ensures that the result is found only if it is inside the same sentence
`[tag="N.*"] [ ] [tag="V.*"] within <s/>`

something shorter **`within`** *something longer*

finds *something shorter* only if it appears inside *something longer*, e.g. adjective *technical* but only if it appears inside a sequence of 3 adjectives
`[lc="technical"]within [tag="J.*"]{3}`

`<s/> containing`

finds sentences which contain something else
`<s/> containing [tag="N.*"] [ ] [tag="V.*"]`

something longer **`containing`** *something shorter*

finds *something longer* only if it contains *something shorter*
`[tag="N.*"][ ]{1,3}[tag="V.*"] containing [lc="often"]`

Only the thing before `within/containing` will be highlighted in red as KWIC. Using the other operator to change the highlighting.

`meet`

finds something (*staff*) only if something else (*member*) is to the left/right
def.attr. **lemma** `(meet "staff" "member" -1 2)`

Structures

`<doc>` `<p>` `<s>` beginning of a document, paragraph, sentence ▪ `</doc>` `</p>` `</s>` end of a structure ▪ `<doc/>` `<p/>` `<s/>` the whole structure

Structures in CQL

`<doc>[]` finds the first token of each document
`[lc="local"] within <doc region="UK"/>` finds the word *local* in documents whose region is UK

full CQL manual online: http://**ske.li/cql**

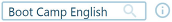
CQL cheat sheet

most frequently used tags in the

English tagset

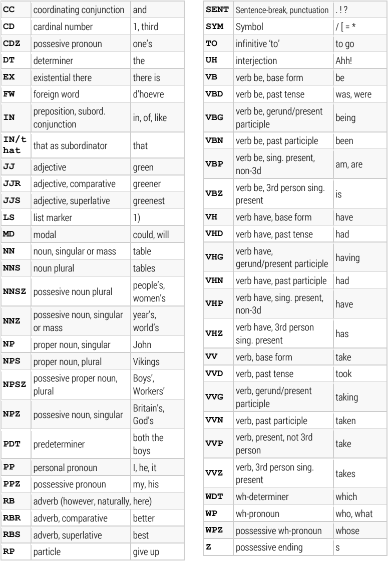
N.*	noun
V.*	verb
J.*	adjective
RB.?	adverb
PP.?	pronoun
CC	conjunction
IN	preposition
DT	determiner
CD	numeral
RP	particle

Click ⓘ to display the tagset of your corpus.



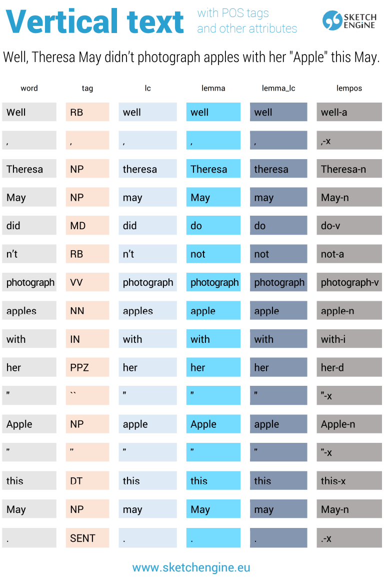
SKETCH ENGINE

Full English tagset



CC	coordinating conjunction	and
CD	cardinal number	1, third
CDZ	possesive pronoun	one's
DT	determiner	the
EX	existential there	there is
FW	foreign word	d'hoevre
IN	preposition, subord. conjunction	in, of, like
IN/that	that as subordinator	that
JJ	adjective	green
JJR	adjective, comparative	greener
JJS	adjective, superlative	greenest
LS	list marker	1)
MD	modal	could, will
NN	noun, singular or mass	table
NNS	noun plural	tables
NNSZ	possesive noun plural	people's, women's
NNZ	possesive noun, singular or mass	year's, world's
NP	proper noun, singular	John
NPS	proper noun, plural	Vikings
NPSZ	possesive proper noun, plural	Boys', Workers'
NPZ	possesive noun, singular	Britain's, God's
PDT	predeterminer	both the boys
PP	personal pronoun	I, he, it
PPZ	possessive pronoun	my, his
RB	adverb (however, naturally, here)	
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	particle	give up

SENT	Sentence-break, punctuation	. ! ?
SYM	Symbol	/ [= *
TO	infinitive 'to'	to go
UH	interjection	Ahh!
VB	verb be, base form	be
VBD	verb be, past tense	was, were
VBG	verb be, gerund/present participle	being
VBN	verb be, past participle	been
VBP	verb be, sing. present, non-3d	am, are
VBZ	verb be, 3rd person sing. present	is
VH	verb have, base form	have
VHD	verb have, past tense	had
VHG	verb have, gerund/present participle	having
VHN	verb have, past participle	had
VHP	verb have, sing. present, non-3d	have
VHZ	verb have, 3rd person sing. present	has
VV	verb, base form	take
VVD	verb, past tense	took
VVG	verb, gerund/present participle	taking
VVN	verb, past participle	taken
VVP	verb, present, not 3rd person	take
VVZ	verb, 3rd person sing. present	takes
WDT	wh-determiner	which
WP	wh-pronoun	who, what
WPZ	possessive wh-pronoun	whose
Z	possessive ending	s



Vertical text

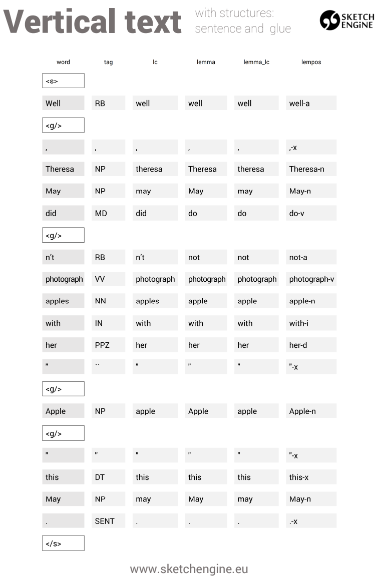
with POS tags and other attributes

SKETCH ENGINE

Well, Theresa May didn't photograph apples with her "Apple" this May.

word	tag	lc	lemma	lemma_lc	lempos
Well	RB	well	well	well	well-a
,	,	,	,	,	,-x
Theresa	NP	theresa	Theresa	theresa	Theresa-n
May	NP	may	May	may	May-n
did	MD	did	do	do	do-v
n't	RB	n't	not	not	not-a
photograph	VV	photograph	photograph	photograph	photograph-v
apples	NN	apples	apple	apple	apple-n
with	IN	with	with	with	with-i
her	PPZ	her	her	her	her-d
"	``	"	"	"	"-x
Apple	NP	apple	Apple	apple	Apple-n
"	"	"	"	"	"-x
this	DT	this	this	this	this-x
May	NP	may	May	may	May-n
.	SENT	.	.	.	.-x

www.sketchengine.eu



Vertical text

with structures: sentence and glue

SKETCH ENGINE

word	tag	lc	lemma	lemma_lc	lempos
<s>					
Well	RB	well	well	well	well-a
<g/>					
,	,	,	,	,	,-x
Theresa	NP	theresa	Theresa	theresa	Theresa-n
May	NP	may	May	may	May-n
did	MD	did	do	do	do-v
<g/>					
n't	RB	n't	not	not	not-a
photograph	VV	photograph	photograph	photograph	photograph-v
apples	NN	apples	apple	apple	apple-n
with	IN	with	with	with	with-i
her	PPZ	her	her	her	her-d
"	``	"	"	"	"-x
<g/>					
Apple	NP	apple	Apple	apple	Apple-n
<g/>					
"	"	"	"	"	"-x
this	DT	this	this	this	this-x
May	NP	may	May	may	May-n
.	SENT	.	.	.	.-x
</s>					

www.sketchengine.eu

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

How to store integer numbers

BNC: 112,345,722 tokens

- whole data 4-byte encoding: 449,382,888 bytes
- whole data delta difference coding: 189 MB
- *the*: frequency 5,415,707 (4.8 %)
- 4-byte integer encoding: 21,662,828 bytes
- delta difference coding: 5,213,473 bytes (24 %)

enTenTen20: 43,125,207,462 tokens

- whole data 4-byte encoding: 172,500,829,848 bytes
- whole data delta difference coding: 75 GB
- *the*: frequency 1,915,064,722 (4.44 %)
- 4-byte integer encoding: 7,660,258,888 bytes
- delta difference coding: 1,877,715,456 bytes (24.5 %)

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`

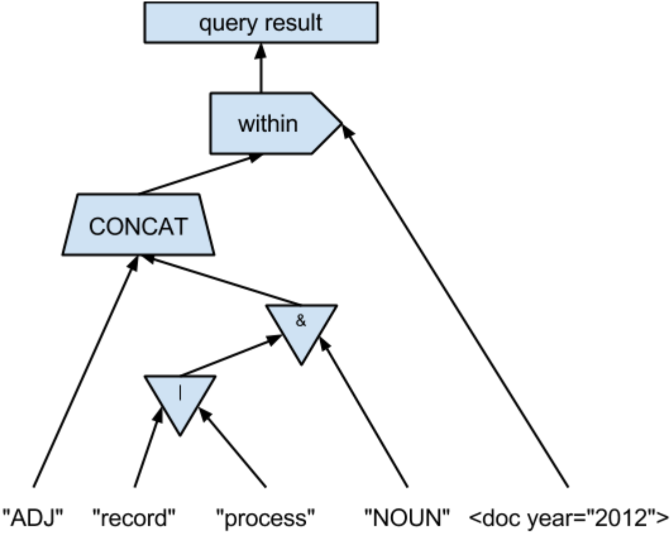
Performance evaluation

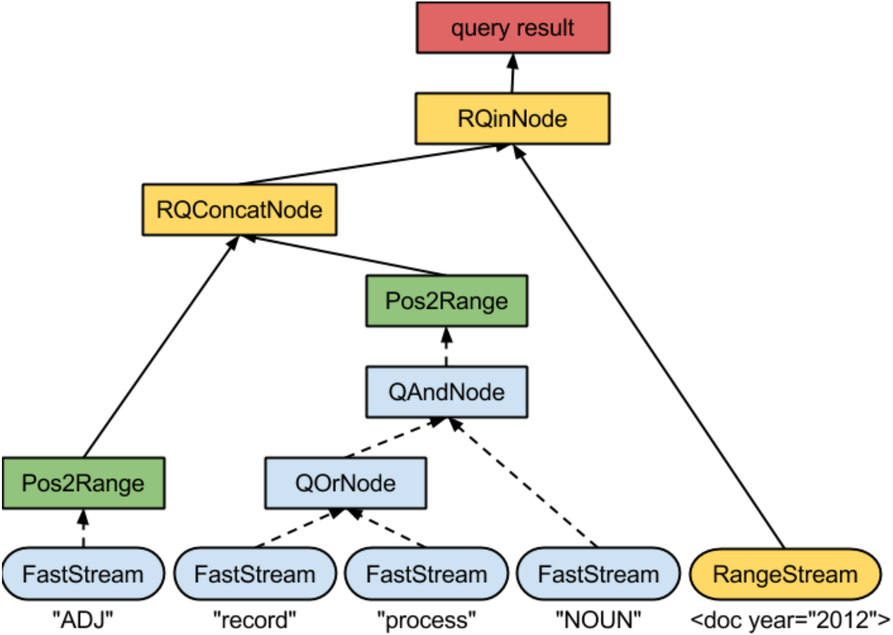
Table: Query performance evaluation – corpora legend: ○ BNC (110M tokens), • BiWeC (version with 9.5G tokens), ∗ Czes (1.2G tokens)

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

Conclusions

- special database management systems for processing text corpora needed
- trade-offs between compile-time and run-time, in-memory and off-memory
- CQL
- Manatee

Assignment