

Corpus Storage

IB047

Pavel Rychlý

pary@fi.muni.cz

March 13, 2023

Pavel Rychlý Corpus Storage

Corpus storage

- enumeration, lexicon
- text, compression
- indexes
- structures
- suffix trees

Pavel Rychlý Corpus Storage

Corpus model

- corpus consists of attributes and structures
- each one has name (= file prefix)
- each one is independent, they are linked by token numbers
- corpus is sequence of tokens, token numbers from 0 to the size of the corpus (-1)
- standard attribute names: word, tag, lemma, lempos

```
<doc id="S001"
      author="Sha">
  <phr type="V">
    to TO
    be VB
  </phr>
  or C
  <phr type="V">
    not NOT
    to TO
    be VB
  </phr>
</doc>
```

Pavel Rychlý Corpus Storage

Enumeration

- using numbers instead of objects
- example: charsets
- corpus:
 - object: value of attributes on each token, value of structures' attributes
 - words, tags, ...
 - each type has its own sequence of numbers

Pavel Rychlý Corpus Storage

Lexicon

enumeration of an attribute values

- basic functions:
 - word → number
 - number → word
- word = character string without any structure
- basic functions:
 - find words matching a regular expression
 - sorting using locales

Pavel Rychlý Corpus Storage

Lexicon implementation

attr.lex ' \0' separated strings
attr.lex.idx array of string starts
attr.lex.srt alphabetically sorted string numbers

```
$ od -t a 2bv-2b/word.lex
0000000  t  o nul  b  e nul  o  r nul  n  o  t nul

$ od -td4 2bv-2b/word.lex.idx
0000000      0      3      6      9

$ od -td4 2bv-2b/word.lex.srt
0000000      1      3      2      0
```

Pavel Rychlý Corpus Storage

Lexicon implementation

Usage

- `lslex` for listing
- `lslex -m` for building `.srt` file
- `lsclex` for listing corpus lexicon (with freqs)

```
$ lslex 2bv-2b/word
0 to
1 be
2 or
3 not
```

```
$ lslex 2bv-2b/word|sort -k2
1 be
3 not
2 or
0 to
```

Storage of corpus text

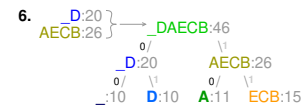
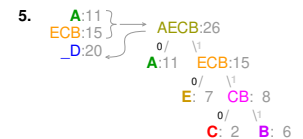
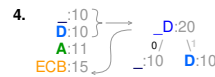
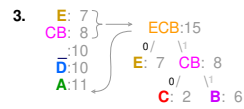
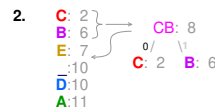
- sequence of positions
- one word ID on each position
- simple storage = array of 32-bit numbers used for attributes of structures
- file `attr.text`
- the biggest part of stored data
→ compression

Compression of corpus text

- codes for each number
- different code length
- shorter codes for more frequent words
- Huffman coding
 - optimal code
 - requires code table

Huffman coding

1. "A_DEAD_DAD_CEDED_A_BAD_BABE_A_BEADED_ABACA_BED"



8. "100001110100100011001101110110011100100010001111100100111110111111001000111110100110011001111101111011101000111111001"

Universal codes

non-parametric, prefix-free, universal

N	unary	binary	gamma	delta
0	1	0000	1	1
1	01	0001	01 0	010 0
2	001	0010	01 1	010 1
3	0001	0011	001 00	011 00
4	00001	0100	001 01	011 01
5	000001	0101	001 10	011 10
6	0000001	0110	001 11	011 11
7	00000001	0111	0001 000	00100 000
8	000000001	1000	0001 001	00100 001
9	0000000001	1001	0001 010	00100 010

Compressed text implementation

`attr.text` delta codes of numbers

`attr.text.seg` starts of a segments (64 positions) in bits

```
$ od -t x1 word.text
00000000 a3 66 69 6e 44 54 00 00 00 00 00 00 00 00 00 00
00000020 88 80 07 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 45 4d 01
```

```
$ od -t d4 word.text.seg
00000000          256          275
```

Indexes

- store hits for each word
- basic functions:
 - word ID → list of positions
 - word ID → number of hits

Pavel Rychlý Corpus Storage

Index implementation

- `attr.rev` lists of positions
delta coding used for position differences
- `attr.rev.idx` array of lists' starts
- `attr.rev.cnt` array of lists' sizes

```
$ od -t x1 word.rev
00000000 a3 66 69 6e 44 52 02 0d 62 00 0a 06

$ od -t d4 word.rev.idx
00000000      7      8      10      11
00000020      12

$ od -t d4 word.rev.cnt
00000000      2      2      1      1
```

Pavel Rychlý Corpus Storage

Structures

- `struct.rng` – array of (begin,end) pairs
- 32-bit or 64-bit (if TYPE is map64 or file64)
- attributes form a “corpus”

```
<doc id="S001"
  author="Sha">
  <phr type="V">
    Main corpus      doc      phr
  to      TO
  be      VB
  </phr>
  or      C
  <phr type="V">
    not      NOT      S001  Sha  V
  to      TO
  be      VB
  </phr>
</doc>
```

Pavel Rychlý Corpus Storage

Structure implementation

```
$ od -t d4 doc.rng
00000000      0      6

$ od -t d4 phr.rng
00000000      0      2      3      6

$ ls phr.*
phr.rng      phr.type.lex.srt  phr.type.rev.idx
phr.type.lex  phr.type.rev     phr.type.text
phr.type.lex.idx  phr.type.rev.cnt

$ od -t d4 phr.type.text
00000000 1852401315 21577      0      0
00000020      0      0
```

Pavel Rychlý Corpus Storage

Sufixové stromy

- suffix tree, position tree, subword tree
- pro řetězec znaků
 - všechny podřetězce končí posledním znakem
 - přidáváme speciální symbol "konec", aby žádný řetězec nebyl prefixem jiného
 - z podřetězců vytvoříme trie (digitální strom)

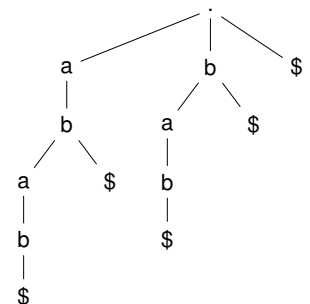
Pavel Rychlý Corpus Storage

Sufixové stromy - příklad

- řetězec: abab\$

- podřetězce:

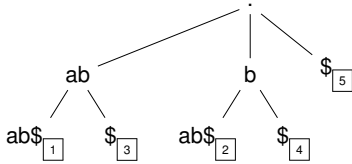
- 1 abab\$
- 2 bab\$
- 3 ab\$
- 4 b\$
- 5 \$



Pavel Rychlý Corpus Storage

Sufixové stromy - vlastnosti

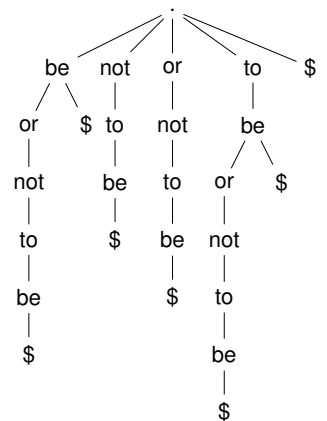
- řetězec délky N
 - N listů
 - v kompaktním trie max. $2 \cdot N$ uzlů
- listy ohodnoceny pozicí
- vnitřní uzly ohodnoceny velikostí podstromu



Pavel Rychlý Corpus Storage

Sufixové stromy pro korpus

- znaky → slova
- řetězec → text korpusu
- funkce
 - strom (seznam) pozic pro každé slovo i posloupnost slov
 - upořádání pozic je lexikografické



Pavel Rychlý Corpus Storage

Sufixová pole

- suffix array, PAT array
- optimální uložení sufixových stromů
- uložení pozic v listech suf. stromu podle pevného uspořádání hran
- stejné funkce jako suf. stromy

abab\$:

1	3	2	4	5
---	---	---	---	---

to be or not to be \$:

1	5	3	2	0	4	6
---	---	---	---	---	---	---

Pavel Rychlý Corpus Storage