

# Corpus Storage

IB047

Pavel Rychlý

pary@fi.muni.cz

March 20, 2017

# Corpus storage

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

# 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>
```

# 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

## enumeration of an attribute values

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

# Lexicon implementation

`attr.lex` '\0' separated strings

`attr.lex.idx` array of string starts

`attr.lex.srt` alphabetically sorted string numbers

## Use

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

# 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



# 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

- store hits for each word
- basic functions:
  - word ID  $\rightarrow$  list of positions
  - word ID  $\rightarrow$  number of hits

# 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

# 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">
```

```
to    TO
```

```
be    VB
```

```
</phr>
```

```
or    C
```

```
<phr type="V">
```

```
not   NOT
```

```
to    TO
```

```
be    VB
```

```
</phr>
```

```
</doc>
```

```
Main corpus
```

```
doc
```

```
phr
```

```
to    TO
```

```
be    VB
```

```
or    C
```

```
not   NOT
```

```
to    TO
```

```
be    VB
```

```
S001
```

```
Sha
```

```
V
```

```
V
```

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

# Sufixové stromy - příklad

■ řetězec: abab\$

■ podřetězce:

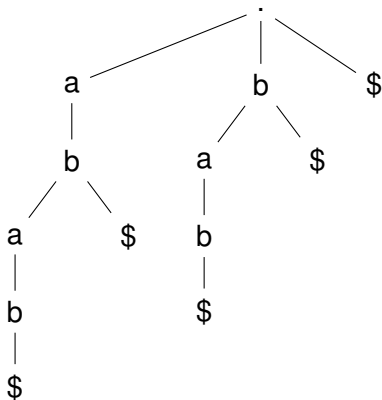
1 abab\$

2 bab\$

3 ab\$

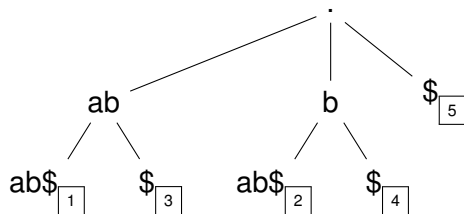
4 b\$

5 \$



# Sufixové stromy - vlastnosti

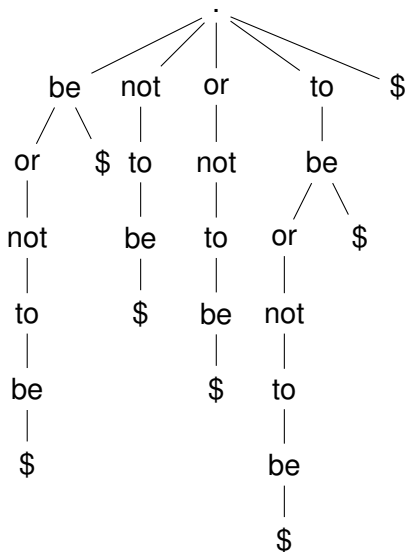
- řetězec délky  $N$ 
  - $N$  listů
  - v kompaktním trie max.  $2 \cdot N$  uzlů
- listy ohodnoceny pozicí
- vnitřní uzly ohodnoceny velikostí podstromu





# Sufixové stromy pro korpus

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



# 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
---	---	---	---	---	---	---