PV030 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2013

Outline (Week seven)

- Excursus to the computational linguistics.
- Corpus linguistics as an TIS example.
- Search methods with preprocessing of text and pattern (query).

Lemmatization for index creation

Morphology utilization for creating of dictionary

- stem/ root of words (učit, uč);
- program ajka (abin), `http://nlp.fi.muni.cz/projekty/ajka/` examples;
- a techniques of patterns for stem determination;

Registry creating – thesaurus

- Thesaurus – a dictionary, containing hierarchical and associative relations and relations of equivalence between particular terms.
- Relations between terms/lemmas:
  - ***synonyms*** – relation to a standard term; e.g. „see“;
  - relation to a related term (RT); e.g. „see also“;
  - relation to a broader term (BT);
  - relation to a narrower term (NT);
  - ***hypernyms*** (car:means of transport); ***hyponyms*** (bird:jay); ***meronym*** (door:lock); ***holonyms*** (hand:body); ***antonyms*** (good:bad).
- Dog/Fík, Havel/president

Thesaurus construction

manually/ half-automatically

- heuristics of thesaurus construction:
  - hierarchical structure/s of thesaurus
  - field thesauri, the semantics is context-dependent (e.g. field, tree in informatics)
  - compounding of terms with a similar frequency
  - exclusion of terms with a high frequency
- breadth of application of thesaurus and lemmatizer: besides of spelling indexing, base of grammar checker, fulltext search.
- projekts WORDNET, EUROWORDNET
- `module add wordnet; wn`
  `wn faculty -over -simsn -coorn`

Hierarchical thesaurus

- Knowledge base creation for exact evaluation of document relevance.
- ***topic*** – processing of semantic maps of terms Visual Thesaurus `http://www.visualthesaurus.com`.
- Tovek Tools, Verity.

Part I

Excursus to the Computational Linguistics

Computational linguistics

- string searching – words are strings of letters.
- word-forming – morphological analysis.
- grammar (CFG, DFG) – syntactic analysis.
- meaning of sentences (TIL) – semantic analysis.
- context – pragmatic analysis.
- full understanding and communication ability – information.

Corpus Query Processor

basic queries

- „Havel“;

  45: Český prezident Václav <Havel> se včera na
  89: jak řekl Václav <Havel> , každý občan
  248: více než rokem <Havel> řekl Pravda vítězí

regular expressions

- „Pravda|pravda“;
- „(P|p)ravda“;
- „(P|p)ravd[a,u,o,y]“;
- „pravd.*“; „pravd.+“; „post?el“;

word sequence

- „prezident(a|u)“ „Havl(a|ovi)“;
- „a tak“;
- „prezident“; []* „Havel“;
- „prezident“ („republiky“ „Vaclav“)? „Havel“;

Corpus Query Processor

queries for positional attributes

- [word = „Havel“];
- [lemma = „prezident“] []* [lemma = „Havel“];
- … ženu prezidenta Havla …
  [lemma = „hnát“] [] [lemma = „Havel“];
- [word = „žen(u|eme)“ & lemma !=„žena“]; | … *or*
  ! … *not*

some other possibilities

- [lemma = „prezident“] []* [lemma = „Havel“] within *s*; …10, 3 *s*
- [lemma = „Havel“] within 20 </*s*>„Pravda“
- <*s*>a:[*word*= „Žena|Muž|Člověk“] []* [lemma = a.lemma]

Face and back of relevant searching

Large computational power of today's computers enables:

- efficient storing of large amount of text data (compression, indexing);
- efficient search for text strings.

A man sitting behind a computer uses all this, to obtain from so processed documents information, that he is interested. Really?

Example: In text database there is stored a few last years of daily newspaper. I'd like to obtain information about president Václav Havel.
a/>HAVEL
b/>more precise queries
c/...
...

Computer (computational power)	+	human (inteligence)	=	valuable information
	+	time		
	+	money		

The goal of everybody → is to transfer the largest possible part of intelligence (time, money, ...) to computer.

Face and back of relevant searching

information	ideal of ideals	no	Searching	
pragmatic analysis	context	no	information	Correct
semantic analysis	sentence meaning TIL	starting-up	Spell	translation
syntactic analysis	grammar CFG, DCG	partially	check	
morphological analysis	word-forming lemma	yes	Check	Simple translat
words are strings of letters	string searching	yes		

Face and back of data acquisition from natural language

Do we really know, what is information contained in the text in natural language?

- František Novák weighs 100 kg. ⟶ **RDB**
  object property value — attribut1, attribut2, ...
- František Novák likes beer. ? key value
  František Novák likes
  Jana Novotná
- F. N. is an old honest man. ⟶ ?
  Spring erupted in full force.

Words of the natural language denote objects, their properties and relations between them. It's possible to see the words and sentences also as „functions" of its kind, defined by their meaning.

- A man, who climbed a highest Czech eight-thousander, is my grandson.

Corpus linguistics

- **Corpus**: electronic collection of texts, often indexed by linguistic tags.
- Corpus as a text information system: corpus linguistics.
- BNC, Penn Treebank, DESAM, PNK, ...; ranges from millions to billion positions (words), special methods necessary.
- Corpus managers CQP, GCQP, Manatee/Bonito, `http://www.fi.muni.cz/~pary/`

see [MAR].

What's a corpus?

Definition: ***Corpus*** is a large, internaly structured compact file of texts in natural language electronically stored and processable.

- Indian languages have no script – for a finding of a grammar it's necessary to write up the spoken word.
- 1967 – 1. corpus in U. S. A. (Kučera, Francis) 1 000 000 words.
- Noam Chomsky – refuses corpora.
- Today – massive expansion.

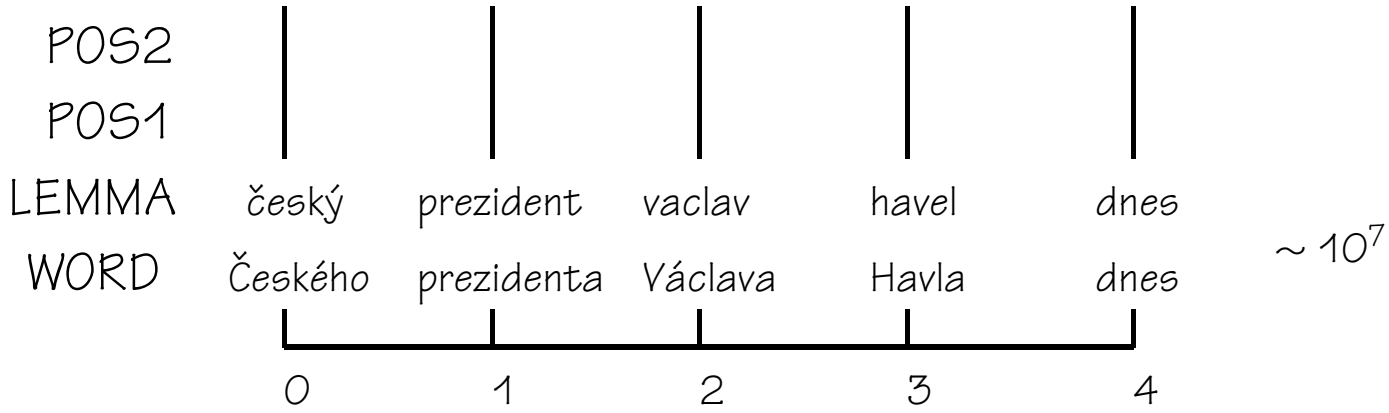
Corpora on FI

- WWW page of Pavel Rychlý (~pary) links to basic information. Bonito, Manatee.
- IMS CORPUS WORKBENCH – a toolkit for efficient representation and querying over large text files.

Logical view of corpus

Sequence of words at numbered positions (first word, nth word), to which ***tags*** are added (addition of tags called corpus ***tagging***). Tags are morphological, grammatical and any other information about a given word. It leads to more general concept of ***position attributes***, those are the most important tagging type. Attributes of this class have a value (string) at every corpus position. To every of them one word is linked as a basic and positional attribute word. In addition to this attribute, further position attributes may be bundled with each position of any text, representing the morphological and other tags.

Structural attributes – sentences, paragraphs, title, article, SGML.



Internal architecture of corpus

Two key terms of internal representation of position attributes are:

- **Uniform representation:** items for all attributes are encoded as integer numbers, where the same values have the same digital code. A sequence of items is then represented as a sequence of integers. Internal representation of attribute word (as well as of any other pos. attribute) is `array(0..p-1) of Integer`, where `p` is position count of corpus.
- **Inverted file:** for a sequence of numbers representing a sequence of values of a given attribute, the inverted file is created. This file contains a set of occurrences in position attribute for every value (better value code). Inverted file is needed for searching, because it directly shows a set of occurrences of a given item, the occurrences then can be counted in one step.

Internal architecture of corpus (cont.)

File with encoded attribute values and inverted file as well have auxiliary files.

- The first data structure is a **list of items** or „lexicon“: it contains a set of different values. Internally it's a set of strings occurring in the sequence of items, where a symbol `Null` (octal 000) is inserted behind every word. The list of items already defines a code for every item, because we suppose the first item in the list to have a code 0, following 1 etc.

Internal architecture of corpus (cont.)

There are three data structures for the inverted file:

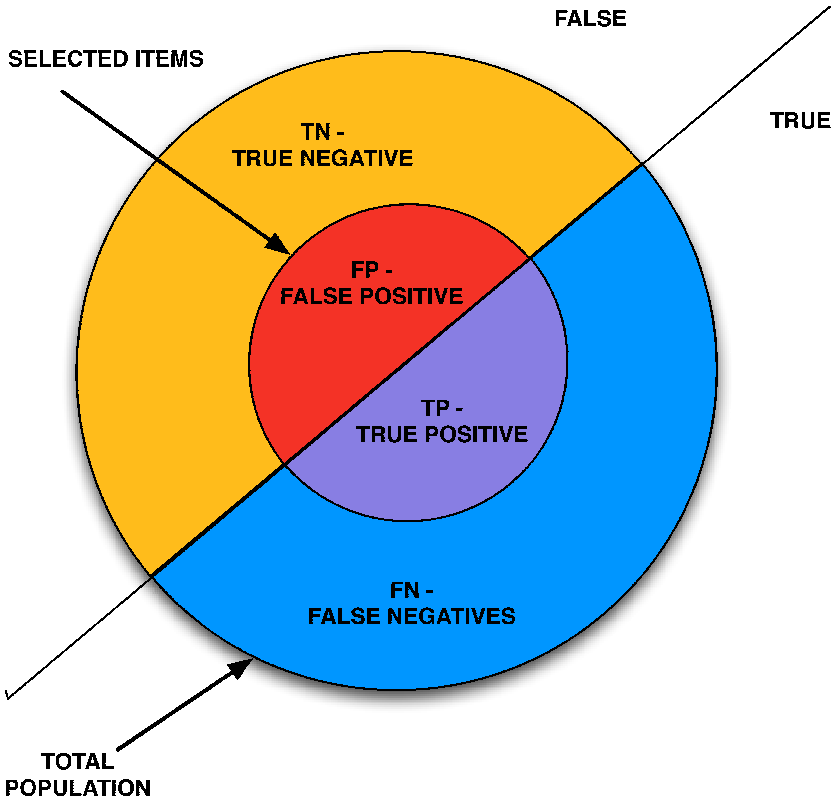
- The first is an independent inverted file, that contains a set of corpus positions.
- The second is an index of this file. This index returns for every code of item an input point belonging to an occurrence in inverted file.
- The third is a table of frequency of item code, which for each item code gives a number of code occurrence in corpus (that is of course the same as the size of occurrence set).

Search methods IV.

Preprocessing of text and pattern (query): overwhelming majority of today's TIS. Types of preprocessing:

- *n*-gram statistics (fragment indexes).
- special algorithms for indexes processing (coding, compression) and relevance evaluation (PageRank Google)
- usage of natural language processing methods (morphology, syntactic analysis, semantic databases) an aggregation of information from multiple sources (systems AnswerBus, START).
- signature methods.

Sensitivity



$$\text{Accuracy} = \frac{tp+tn}{tp+fp+fn+tn}$$

$$\text{Recall (Sensitivity)} = \frac{tp}{tp+fn}$$

$$\text{Precision (Positive Predictive Value)} = \frac{tp}{tp+fp}$$

$$\text{False Positive Rate} = \frac{fp}{fp+tn}$$

$$\text{False Negative Rate} = \frac{fn}{tp+fn}$$

$$\text{Specificity} = \frac{tn}{tn+fp}$$

$$\text{Negative Predictive Value} = \frac{tn}{tn+fn}$$

Relevance

Definition: ***Relevance*** (of answers to a query) is a rate range, by which a selected document coincides with requirements imposed on it.
Ideal answer $\equiv$ real answer
Definition: ***Coefficient of completeness (recall)*** $R = \frac{m}{n}$, where m is a count of selected relevant records and n is a count of all relevant records in TIS.
Definition: ***Coefficient of precision*** $P = \frac{m}{o}$, where o is count of all selected records by a query.
We want to achieve maximum R and P, tradeoff.

Standard values: 80 % for P, 20 % for R.
Combination of completeness and precision:
coefficient $F_b = \frac{(b^2+1)PR}{b^2P+R}$. ($F_0 = P$, $F_\infty = R$, where $F_1 = F$ P and R weighted equally).

Fragment index

- The fragment ybd is in English only in the word molybdenum.
- Advantages: fixed dictionary, no problems with updates.
- Disadvantages: language dependency and thematic area, decreased precision of search.

Outline (Week seven)

- Google as an example of web-scale information system.
- Jeff Dean's video – historical notes of Google search developments.
- Google – system architecture.
- Google – PageRank.
- Google File System.
- Implementation of index systems

Gooooooooooooogle – a bit of history

An example of anatomy of global (hyper)text information system (www.google.com).

- 1997: google.stanford.edu, students Page and Brin
- 1998: one of few quality search engines, whose basic fundamentals and architecture (or at least their principles) are known – therefore a more detailed analysis according to the article [G00]
  `http://www7.conf.au/programme/fullpapers/1921com1921.htm`.
- 2012: clear leader in global web search

Gooooooooooooogle – anatomy

- Several innovative concepts: PageRank, storing of local compressed archive, calculation of relevance from texts of hypertext links, PDF indexing and other formats, Google File System, Google Link...
- The system anatomy. see [MAR]

Google: Relevance

The crucial thing is documents' relevance (credit) computation.

- ☞ Usage of tags of text and web typography for the relevance calculation of document terms.
- ☞ Usage of text of hyperlink is referring to the document.

Google: PageRank

- **PageRank**: objective measure of page importance based on citation analysis (suitable for ordering of answers for queries, namely page relevance computation).
- Let pages $T_1,\ldots,T_n$ (citations) point to a page A, total sum of pages is m. PageRank

$$PR(A) = \frac{(1-d)}{m} + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots \frac{PR(T_n)}{C(T_n)}\right)$$

- PageRank can be calculated by a simple iterative algorithm (for tens of millions of pages in hours on a normal PC).
- PageRank is a probability distribution over web pages.
- PageRank is not the only applied factor, but coefficient of more factors. A motivation with a random surfer, dumping factor d, usually around 0.85.

Data structures of Google

- Storing of file signatures
- Storing of lexicon
- Storing of hit list.
- Google File System

Index system implementation

- Inverted file – indexing file with a bit vector.
- Usage of document list to every key word.
- Coordinate system with pointers [MEL, fig. 4.18, page 46].
- Indexing of corpus texts: Finlib
  `http://www.fi.muni.cz/~pary/dis.pdf` see [MAR].
- Use of Elias coding for a compression of hit list.

Index system implementation (cont.)

- Efficient storing of index/dictionary [lemmas]: ***packed trie***, Patricia tree, and other tree structures.
- Syntactic neural network (S. M. Lucas: Rapid best-first retrieval from massive dictionaries, Pattern Recognition Letters 17, p. 1507–1512, 1996).
- Commercial implementations: Verity engine, most of web search engines – with few exceptions – hide their key to success.

Dictionary representation by FA I

Article M. Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing see [MAR]

- Dictionary representation by finite automaton.
- Ambiguities, unification of minimized deterministic automata.
- Example: done,do.V3:PP
  done,done.A0
- Morphological dictionary as a list of pairs [word form, lemma].
- Compaction of storing of data structure of automata (Liang, 1983).
- Compression ratio up to 1:20 in the linear approach (given the length of word).

Dictionary representation by FA II

- Transducer for dictionary representation.
- Deterministic transducer with 1 output (subsequential transducer) for dictionary representation including one string on output (information about morphology, hyphenation,...).
- Deterministic transducer with p outputs (p–subsequential transducer) for dictionary representation including more strings on output (ambiguities).
- Determinization of the transducer generally unrealizable (the class of deterministic transducers with an output is a proper subclass of nondeterministic transducers); for purposes of natural language processing, though, usually doesn't occur (there aren't cycles).

Dictionary representation by FA III

- An addition of a state to a transducer corresponding (w_1,w_2) without breaking the deterministic property: first a state for (w_1,ε), then with resulting state final state with output w_2.
- Efficient method, quick, however not minimal; there are minimizing algorithms, that lead to spatially economical solutions.
- Procedure: splitting of dictionary, creation of det. transducers with p outputs, their minimization, then a deterministic unification of transducers and minimizing the resulting.
- Another use also for the efficient indexing, speech recognition, etc.

Part II

Coding

Outline (Week seven)

- Coding.
- Entropy, redundancy.
- Universal coding of the integers.
- Huffman coding.
- Adaptive Huffman coding.

Coding – basic concepts

Definition: **Alphabet** A is a finite nonempty set of symbols.
Definition: **Word** (**string**, **message**) over A is a sequence of symbols from A.
Definition: **Empty string** ε is an empty sequence of symbols. A set of nonempty words over A is labeled A^+.
Definition: **Code** K is a triad (S, C, f), where S is finite set of **source units**, C is finite set of **code units**, $f : S \longrightarrow C^+$ is an injective mapping. f can be expanded to $S^+ \longrightarrow C^+$: $F(S_1S_2 \ldots S_k) = f(S_1)f(S_2) \ldots f(S_k)$. C^+ is sometimes called **code**.

Basic properties of the code

Definition: $x \in C^+$ is ***uniquely decodable*** regarding f, if there is maximum one sequence $y \in S^+$ so, that $f(y) = x$.

Definition: Code $K = (S, C, f)$ is ***uniquely decodable*** if all strings in C^+ are uniquely decodable.

Definition: A code is called a ***prefix*** one, if no code word is a prefix of another.

Definition: A code is called a ***suffix*** one, if no code word is a suffix of another.

Definition: A code is called a ***affix*** one, if it is prefix and suffix code.

Definition: A code is called a ***full*** one, if after adding of any additional code word a code arises, that isn't uniquely decodable.

Basic properties of code

Definition: ***Block code of length*** n is such a *code*, in which all *code words* have length n.

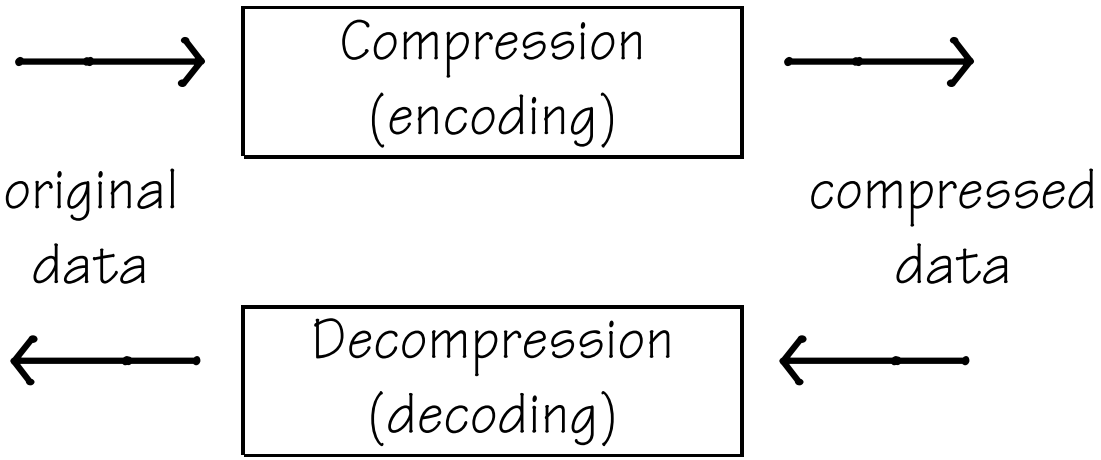
Example: block ? prefix

block $\Rightarrow$ prefix, but not vice versa.

Definition: A *code* $K = (S, C, f)$ is called ***binary***, if $|C| = 2$.

Compression and decompression

Definition: ***Compression*** (***coding***), ***decompression*** (***decoding***):



Definition: ***Compression ratio*** is a ratio of length of compressed data and length of original data.

Example: Suggest a binary prefix code for decimal digits, if there are often numbers 3 a 4, and rarely 5 and 6.

Entropy and redundancy I

Let Y be a random variable with a probability distribution $p(y) = P(Y = y)$. Then the mathematical expectation (mean rate) $E(Y) = \sum_{y \in Y} yp(y)$.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of source units and let the occurrence probability of unit x_i in information source **S** is p_i for $i = 1, \ldots, n$, $n \in N$.

Definition: ***Entropy of information content of unit*** x_i (measure of amount of information or uncertainty) is $H(x_i) = H_i = -\log_2 p_i$ bits.

A source unit with more probability bears less information.

Entropy and redundancy II

Definition: ***Entropy of information source*** $\mathbf{S}$ is $H(\mathbf{S}) = -\sum_{i=1}^{n} p_i \log_2 p_i$ bits.

True, that $H(\mathbf{S}) = \sum_{y \in Y} p(y) \log \frac{1}{p(y)} = E\left(\log \frac{1}{p(Y)}\right)$.

Definition: ***Entropy of source message*** $X = x_{i_1} x_{i_2} \ldots x_{i_k} \in S^+$ ***of information source*** $\mathbf{S}$ is $H(X, \mathbf{S}) = H(X) = \sum_{j=1}^{k} H_i = -\sum_{j=1}^{k} \log_2 p_{i_j}$ bits.

Definition: ***Length*** $l(X)$ ***of encoded message*** X

$l(X) = \sum_{j=1}^{k} |f(x_{i_j})| = \sum_{j=1}^{k} d_{i_j}$ bits.

Theorem: $l(X) \geq H(X, \mathbf{S})$.

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see `ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kodovani.ps`

Definition: $R(X) = I(X) - H(X) = \sum_{j=1}^{k}(d_{i_j} + \log_2 p_{i_j})$ is ***redundancy of code*** K ***for message*** X.

Definition: ***Average length of code word*** K is $AL(K) = \sum_{i=1}^{n} p_i d_i$ bits.

Definition: ***Average length of source*** $\mathbf{S}$ is

$AE(\mathbf{S}) = \sum_{i=1}^{n} p_i H_i = -\sum_{i=1}^{n} p_i \log_2 p_i$ bits.

Definition: ***Average redundancy of code*** K is

$AR(K) = AL(K) - AE(\mathbf{S}) = \sum_{i=1}^{n} p_i(d_i + \log_2 p_i)$ bits.

Entropy and redundacy IV

Definition: A *code* is an ***optimal*** one, if it has minimal redundancy.
Definition: A *code* is an ***asymptotically optimal***, if for a given distribution of probabilities the ratio $AL(K)/AE(S)$ is close to 1, while the entropy is close to ∞.
Definition: A *code* K is a ***universal*** one, if there are $c_1, c_2 \in R$ so, that average length of code word $AL(K) \leq c_1 \times AE + c_2$.
Theorem: Universal *code* is ***asymptotically optimal***, if $c_1 = 1$.

Universal coding of integers

Definition: **Fibonacci sequence of order** m

$F_n = F_{n-m} + F_{n-m+1} + \ldots + F_{n-1}$ for $n \geq 1$.

Example: F of order 2: $F_{-1} = 0$,, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$,...

Example: F of order 3: $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 7$, $F_5 = 13$,...

Example: F of order 4: $F_{-3} = 0$, $F_{-2} = 0$, $F_{-1} = 0$, $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, $F_4 = 8$, $F_5 = 15$,...

Definition: **Fibonacci representation** $R(N) = \sum_{i=1}^{k} d_i F_i$, where $d_i \in \{0, 1\}$, $d_k = 1$

Theorem: Fibonacci representation is ambiguous, however there is such a one, that has at most $m - 1$ consecutive ones in a sequence d_i.

Fibonacci codes

Definition: ***Fibonacci code of order*** m $FK_m(N) = d_1 d_2 \dots d_k \underbrace{1 \dots 1}_{m-1 \text{ krát}}$,

where d_i are coefficients from previous sentence (ones end a word).

Example: $R(32) = 0*1 + 0*2 + 1*3 + 0*5 + 1*8 + 0*13 + 1*21$, thus $F(32) = 001010011$.

Theorem: $FK(2)$ is a prefix, universal code with $c_1 = 2$, $c_2 = 3$, thus it isn't asymptotically optimal.

The universal coding of the integers II

- unary *code* $a(N) = \underbrace{00\ldots01}_{N-1}$.
- binary *code* $\beta(1) = 1, \beta(2N + j) = \beta(N)j,\ j = 0, 1$.
- β is not uniquely *decodable* (it isn't prefix *code*).
- ternary $\tau(N) = \beta(N)\#$.
- $\beta'(1) = \epsilon, \beta'(2N) = \beta'(N)0, \beta'(2N + 1) = \beta'(N)1, \tau'(N) = \beta'(N)\#$.
- γ: every bit $\beta'(N)$ is inserted between a pair from $a(|\beta(N)|)$.
- example: $\gamma(6) = 0\overline{1}0\overline{0}1$
- $C_\gamma = \{\gamma(N) : N > 0\} = (0\{0, 1\})^*1$ is regular and therefore it's *decodable* by finite automaton.

The universal coding of the integers III

- $\gamma'(N) = a(|\beta(N)|)\beta'(N)$ the same length (bit permutation $\gamma(N)$), but more readable
- $C_{\gamma'} = \{\gamma'(N) : N > 0\} = \{0^k 1\{0,1\}^k : k \geq 0\}$ is not regular and the decoder needs a counter
- $\delta(N) = \gamma(|\beta(N)|)\beta'(N)$
- example: $\delta(4) = \gamma(3)00 = 01100$
- decoder δ: $\delta(?) = 0011?$
- ω:

  $K := 0;$
  while $\lfloor log_2(N) \rfloor > 0$ **do**
  begin $K := \beta(N)K;$
  $N := \lfloor log_2(N) \rfloor$
  end.

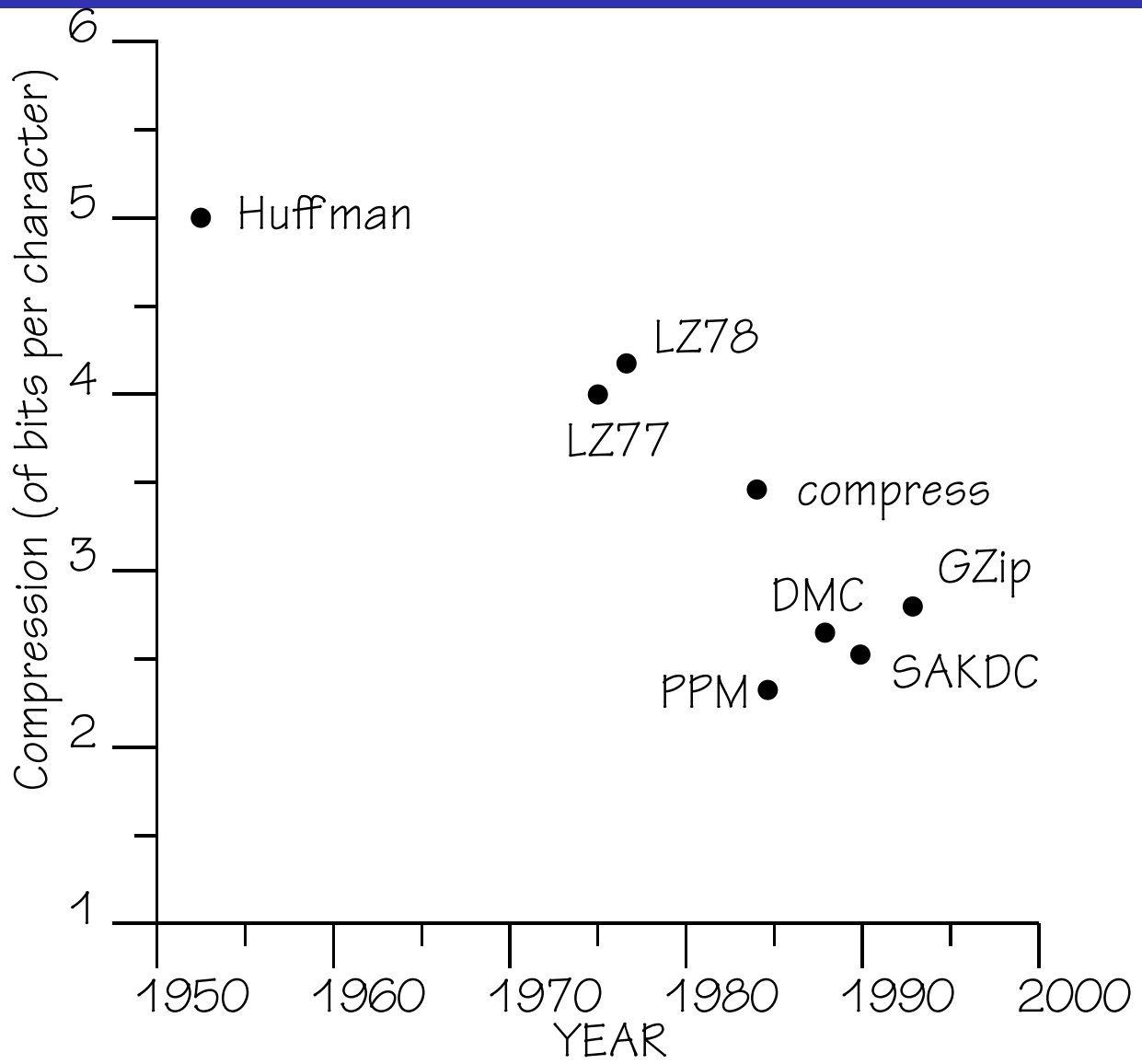
Data compression – introduction

- Information encoding for communication purposes.
- Despite tumultuous evolution of capacities for data storage, there is still a lack of space, or access to compressed data saves time. Redundancy $\longrightarrow$ a construction of a minimal redundant code.
- Data model:
  - structure – a set of units to compression + context of occurrences;
  - parameters – occurrence probability of particular units.
  - data model creation;
  - the actual encoding.

Data compression – evolution

- 1838 Morse, *code e by frequency.*
- 1949 Shannon, Fano, Weaver.
- 1952 Huffman; *5 bits per character.*
- 1979 Ziv-Lempel; **compress** (Roden, Welsh, Bell, Knuth, Miller, Wegman, Fiala, Green, . . .); *4 bits per character.*
- *eighties and nineties* PPM, DMC, **gzip** (zlib), SAKDC; *2–3 bits/character*
- *at the turn of millenium* **bzip2**; *2 bits per character.*
- . . . ?

Evolution of compression algorithms



Prediction and modeling

- redundancy (non-uniform probability of source unit occurrences)
- encoder, decoder, model
- statistical modeling (the model doesn't depend on concrete data)
- semiadaptive modeling (the model depends on data, 2 passes, necessity of model transfer)
- adaptive modeling (only one pass, the model is created dynamically by both encoder and decoder)

Prediction and modeling

- models of order 0 – probabilities of isolated source units (e.g. Morse, character e)
- models with a finite context – Markov models, models of order n (e.g. Bach), $P(a|x_1 x_2 \ldots x_n)$
- models based on finite automata
  - synchronization string, nonsynchronization string
  - automaton with a finite context
  - suitable for regular languages, unsuitable for context-free languages, $P(a|q_i)$

Outline (Week seven)

- Huffman coding.
- Adaptive Huffman coding.
- Aritmetic coding.
- Dictionary methods.
- Signature methods.
- Similarity of documents.
- Compression using neural networks.

Statistical compression methods I

Character techniques

- null suppression – replacement of repetition ≥ 2 of character null, 255, special character S_c
- run-length encoding (RLE) – $S_c X C_c$ generalization to any repetitious character \$ * * * * * *55 $\longrightarrow$ \$$S_c$ * 655
- MNP Class 5 RLE – CXXX *DDDDDBBAAAA* $\longrightarrow$ *5DDDBB4AAA*
- half-byte packing, (EBCDIC, ASCII) SI, SO
- diatomic encoding; replacement of character pairs with one character.
- Byte Pair Encoding, BPE (Gage, 1994)
- pattern substitution
- Gilbert Held: Data & Image Compression

Statistical compression methods II

- Shannon-Fano, 1949, model of order 0,
- code words of length $\lfloor -\log_2 p_i \rfloor$ or $\lfloor -\log_2 p_i + 1 \rfloor$
- $AE \leq AL \leq AE + 1$.
- code tree (2,2,2,2,4,4,8).
- generally it is not optimal, two passes of encoder through text, static $\longrightarrow$x

Shannon-Fano coding

Input: *a sequence of n source units $S[i]$, $1 \leq i \leq n$, in order of nondecreasing probabilities.*
Output: *n binary code words.*

```
begin assign to all code words an empty string;
        SF-SPLIT(S)
end

procedure SF-SPLIT(S);
begin if |S| ≥ 2 then
        begin divide S to sequences S1 and S2 so, that both
                  sequences have roughly the same total probability;
                add to all code words from S1 0;
                add to all code words from S2 1;
                SF-SPLIT(S1); SF-SPLIT(S2);
        end
end
```

Outline (Week seven)

- Agenda (HW 3, corrections and extension of subject materials).
- Universal coding of the integers – completion.
- Data compression, introduction.
- Statistical methods of data compression (Shannon-Fano, Huffman).
- Arithmetic coding.

Home work PV030/2003 nr. 3

A): Calculate $\omega(100)$.

B): Let's have a text : *strč prst skrz krk*.

- ☞ Encode using Shannon-Fano coding and draw a code tree. kódový strom.
- ☞ Calculate a length of an encoded message, it's entropy and redundancy.
- ☞ Calculate an average entropy of source units, and an average length of code word.

C): Encode by arithmetic coding a text *baaba* if you know, that count of *a* is three times higher than *b*.

Huffman coding

- Huffman coding, 1952.
- static and dynamic variants.
- $AEPL = \sum_{i=1}^{n} d[i]p[i]$.
- optimal code (not the only possible).
- $O(n)$ assuming ordination of source units.
- stable distribution $\longrightarrow$ preparation in advance.

Example: (2,2,2,2,4,4,8)

Huffman coding – sibling property

Definition: Binary tree have a ***sibling property*** if and only if

1. each node except the root has a sibling,
2. nodes can be arranged in order of nondecreasing sequence so, that each node (except the root) adjacent in the list with another node, is his sibling (the left sons are on the odd positions in the list and the right ones on even).

Huffman coding – properties of Huffman trees

Theorem: A binary prefix code is a Huffman one $\Leftrightarrow$ it has the sibling property.

- $2n - 1$ nodes, max. $2n - 1$ possibilities,
- optimal binary prefix code, that is not the Huffman one.
- $AR(X) \leq p_n + 0{,}086$, p_n maximum probability of source unit.
- Huffman is a full code, (poor error detection).
- possible to extend to an **affix code**, KWIC, left and right context, searching for $*X*$.

Adaptive Huffman coding

- FGK (Faller, Gallager, Knuth)
- suppression of the past by coefficient of forgetting, rounding, 1, r, r^2, r^n.
- linear time of coding and decoding regarding the word length.
- $AL_{HD} \leq 2AL_{HS}$.
- Vitter $AL_{HD} \leq AL_{HS} + 1$.
- implementation details, tree representation code tables.

Principle of arithmetic coding

- generalization of Huffman coding (probabilities of source units needn't be negative powers of two).
- order of source units; **Cumulative probability** $cp_i = \sum_{j=1}^{i-1} p_j$ source units x_i with probability p_i.
- Advantages:
  - any proximity to entropy.
  - adaptability is possible.
  - speed.

Dictionary methods of data compression

Definition: ***Dictionary*** is a pair $D = (M, C)$, where M is a finite set of words of source language, C mapping M to the set of code words.
Definition: $L(m)$ denotes the length of code word $C(m)$ in bits, for $m \in M$.
Selection of source units:

- static (agreement on the dictionary in advance)
- semiadaptive (necessary two passes trough text)
- adaptive

Statical dictionary methods

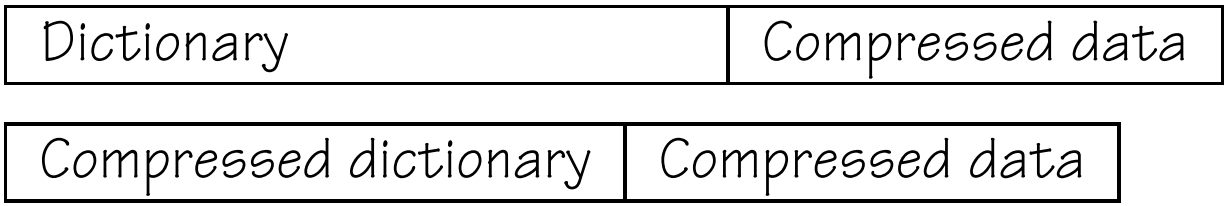
Source unit of the length n – n-grams
Most often bigrams ($n = 2$)

- n fixed
- n variable (by frequency of occurrence)
- adaptive

(50 % of an English text consits of about 150 most frequent words)
Disadvantages:

- they are unable to react to the probability distribution of compressed data
- pre-prepared dictionary

Semiadaptive dictionary methods



Advantages: extensive date (the dictionary is a small part of data – corpora; CQP).

Semiadaptive dictionary methods – dictionary creation procedure

1. The frequency of N-grams is determined for $N = 1, 2, \ldots$.
2. The dictionary is initialized by unigram insertion.
3. N-grams with the highest frequency are gradually added to the dictionary. During K-gram insertion frequencies decrease for it's components of $(K-1)$-grams, $(K-2)$-grams .... If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

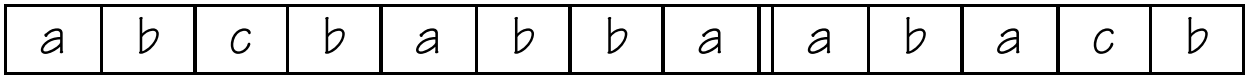
Outline (Week seven)

- Adaptive dictionary methods with dictionary restructuring.
- Syntactic methods.
- Checking of text correctness.
- Querying and TIS models.
- Vector model of documents
- Automatic text structuring.
- Document similarity.

Adaptive dictionary methods

LZ77 – siliding window methods

LZ78 – methods of increasing dictionary



encoded part (window, $N \leq 8192$)

not enc. part ($|B|$ ~10–20 b)

In the encoded part the longest prefix P of a string in not encoded part is searched. If such a string is found, then P is encoded using (I, J, A), where I is a distance of first character S from the border, J is a length of the string S and A is a first character behind the prefix P. The window is shifted by $J + 1$ characters right. If the substring S wasn't found, then a triple $(0, 0, A)$ is created, where A is a first character of not encoded part.

LZR (Rodeh)

$|M| = (N - B) \times B \times t$, t size of alphabet
$L(m) = \lceil \log_2(N - B) \rceil + \lceil \log_2 B \rceil + \lceil \log_2 t \rceil$
Advantage: the search of the longest prefix [KMP]

- LZR uses a tree containing all the prefixes in the yet encoded part.
- The whole encoded yet encoded part is used as a dictionary.
- Because the i in (i, j, a) can be large, the Elias code for coding of the integers is used.

Disadvantage: a growth of the tree size without any limitation $\Rightarrow$ after exceeding of defined memory it's deleted and the construction starts from the beginning.

LZSS (Bell, Storer, Szymanski)

The code is a sequence of pointers and characters. The pointer (i, j) needs a memory as p characters $\Rightarrow$ a pointer only, when it pays off, but there is a bit needed to distinguish a character from a pointer. The count of dictionary items is $|M| = t + (N - B) \times (B - p)$ (considering only substrings longer than p). The bit count to encode is

- $L(m) = 1 + \lceil \log_2 t \rceil$ for $m \in T$
- $L(m) = 1 + \lceil \log_2 N \rceil + \lceil \log_2 (B - p) \rceil$ otherways.

(The length d of substring can be represented as $B - p$).

LZB (Bell), LZH (Brent)

A pointer (i, j) (analogy to LZSS)
If

- the window is not full (at the beginning) and
- the compressed text is shorter than N,

the usage of $\log_2 N$ bytes for encoding of i is a waste. LZB uses phasing for binary coding. – prefix code with increasing count of bits for increasing values of numbers. Elias code γ.
LZSS, where for pointer encoding the Huffman coding is used (i.e. by distribution of their probabilities $\Rightarrow$ 2 throughpasses)

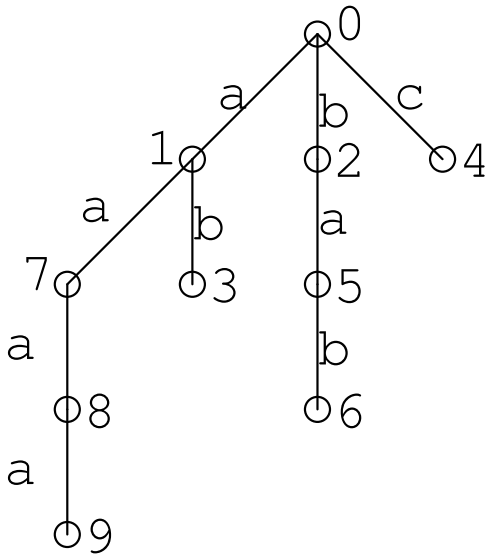
Methods with increasing dictionary

The main idea: the dictionary contains phrases. A new phrase so, that an already existing phrase is extended by a symbol. A phrase is encoded by an index of the prefix and by the added symbol.

LZ78 – example

Input	a	b	ab	c	ba	...
Index	1	2	3	4	5	
Output	(0,a)	(0,b)	(1,b)	(0,c)	(2,a)	

...	Input	bab	aa	aaa	aaaa
	Index	6	7	8	9
	Output	(5,b)	(1,a)	(7,a)	(8,a)



LZFG (Fiala, Green)

A dictionary is stored in a tree structure, edges are labeled with strings of characters. These strings are in the window and each node of the tree contains a pointer to the window and identifying symbols on the path from the root to the node.

LZW (Welch), LZC

The output indexes are only, or

- the dictionary is initiated by items for all input symbols
- the last symbol of each phrase is the first symbol of the following phrase.

Input	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a
Index		4	5		6	7		8			9	10			
Output	1	2		4	3		5			8	1		10	11	

Overflow $\Rightarrow$ next phrase is not transmitted and coding continues statically.
it's a LZW +

- Pointers are encoded with prolonging length.
- Once the compression ratio will decrease, dictionary will be deleted and it starts from the beginning.

LZT, LZMW, LZJ

As LZC, but when a dictionary overflows, phrases, that were least used in the recent past, are excluded from the dictionary. It uses phrasing for binary coding of phrase indexes.
As LZT, but a new phrase isn't created by one character addition to the previous phrase, but the new phrase is constructed by concatenation of two last encoded ones.
Another principle of dictionary construction.

- At the beginning only the single symbols are inserted.
- Dictionary is stored in a tree and contains all the substrings processed by string of the length up to h.
- Full dictionary $\Rightarrow$
  - statical procedure,
  - omitting of nodes with low usage frequency.

Dictionary methods with dictionary restructuring

- Ongoing organization of source units $\longrightarrow$ shorter strings of the code.
- Variants of heuristics (count of occurrences, moving to the beginning (BSTW), change the previous, transfer of X forward).
- BSTW (advantage: high locality of occurrences of a small number of source units.
- Example: I'm not going to the forest, ..., $1^n 2^n k^n$.
- Generalization: ***recency coefficient***, ***Interval coding***.

Interval coding

Representation of the word by total sum of words from the last occurrence.
The dictionary contains words $a_1, a_2, \ldots, a_n$, input sequence contains $x_1, x2, \ldots, x_m$. The value LAST(a_i) containing the interval form last occurrence is initialized to zero.

for $t := 1$ to m **do**
begin $\{x_t = a_i\}$
    if LAST$(x_t = 0)$ **then** $y(t) = t + i - 1$
        else $y(t) = t -$ LAST(x_t);
    LAST(x_t):=t
end .

Sequence $y_1, y_2, \ldots, y_m$ is an output of encoder and can be encoded by one code of variable length.

Syntactical methods

- the grammar of the message language is known.
- left partition of derivational tree of string.
- global numbering of rules.
- local numbering of rules.

- Decision-making states of LR analyzer are encoded.

Context modeling

- fixed context – model of order N.
- combined approach – contexts of various length.
- $p(x) = \sum_{n=0}^{m} w_n p_n(x)$.
- w_n fixed, variable.
- time and memory consuming.
- assignment of probability to the new source unit: $e = \frac{1}{C_n+1}$.
- automata with a finite context.
- dynamic Markov modeling.

Checking the correctness of the text

- Checking of text using frequency dictionary.
- Checking of text using a double frequency dictionary.
- Interactive control of text (ispell).
- Checking of text based on regularity of words, ***weirdness coefficient***.

Weirdness coefficient

Weirdness coefficient of trigram xyz

$KPT = [\log(f(xy) - 1) + \log(f(yz) - 1)]/2 - \log(f(xyz) - 1)$, where $f(xy)$ resp. $f(xyz)$ are relative frequencies of bigram resp. trigram, $\log(0)$ is defined as -10.

Weirdness coefficient of word $KPS = \sqrt{\sum_{i=1}^{n}(KPT_i - SKPT^2)}$, where KPT_i is a weirdness coefficient of i-th trigram $SKPT$ is a mean rate of weirdness coefficients of all trigrams contained in the word.

Outline (Week seven)

- Querying and TIS models.
- Boolean model of documents.
- Vector model of documents.
- TIS Architecture.
- Signature methods.
- Similarity of documents.
- Vector model of documents (completion).
- Extended boolean model.
- Probability model.
- Model of document clusters.
- TIS Architecture.
- Automatic text structuring.
- Documents similarity.
- Lexicon storage.
- Signature methods.

Querying and TIS models

Different methods of hierarchization and document storage ⟶ different possibilities and efficiency of querying.

- ☞ Boolean model, SQL.
- ☞ Vector model.
- ☞ Extended boolean types.
- ☞ Probability model.
- ☞ Model of document clusters.

Blair's query tuning

The search lies in reducing of uncertainty of a question.

1. We find a document with high relevance.
2. We start to query with it's key words.
3. We remove descriptors, or replace them with disjunctions.

Infomap – attempt to semantic querying

System **http://infomap.stanford.edu** – for working with searched meaning/concept (as opposed to mere strings of characters).
Right query formulation is the half of the answer. The search lies in determination of semantically closest terms.

Boolean model

- Fifties: representation of documents using sets of terms and querying based on evaluation of boolean expressions.
- Query expression: inductively from primitives:
  - term
  - attribute_name = attribute_value (comparison)
  - function_name(term) (application of function)

  and also using parentheses and logical conjunctions X and Y, X or Y, X xor Y, not Y.
- disjunctive normal form, conjunctive normal form
- proximity operators
- regular expressions
- thesaurus usage

Languages for searching – SQL

- boolean operators ***and***, ***or***, ***xor***, ***not***.
- positional operators ***adj***, ***(n) words***, ***with***, ***same***, ***syn***.
- SQL extension: operations/queries with use of thesaurus

BT(A)	Broader term
NT(A)	Narrower term
PT(A)	Preferred term
SYN(A)	Synonyms of the term A
RT(A)	Related term
TT(A)	Top term

Querying – SQL examples

```
ORACLE SQL*TEXTRETRIEVAL
SELECT specification_of_items
FROM specification_of_tables
WHERE item
CONTAINS textov_expression
```

Example:

```
SELECT TITLE
FROM BOOK
WHERE ABSTRACT
CONTAINS 'TEXT' AND RT(RETRIEVAL)
'string' 'string'*  *'string' 'st?ing'
'str%ing' 'stringa' (m,n) 'stringb'
'multiword phrases' BT('string',n)
BT('string',*) NT('string',n)
```

Querying – SQL examples

Example:

```
SELECT NAME
FROM EMPLOYEE
WHERE EDUCATION
CONTAINS  RT(UNIVERSITA)
AND LANGUAGES
CONTAINS 'ENGLISH' AND 'GERMAN'
AND PUBLICATIONS
CONTAINS 'BOOK' OR NT('BOOK',*)
```

Stiles technique/ association factor

$$asoc(Q_A, Q_B) = \log_{10} \frac{(fN - AB - N/2)^2 N}{AB(N - A)(N - B)}$$

A – number of documents „hit“ by the query Q_A

B – number of documents „hit“ by the query Q_B (its relevance we count)

f – number of documents „hit“ by both the queries

N – total sum of documents in TIS

cutoff (relevant/ irrelevant)

clustering*/*nesting 1. generation, 2. generation, ...

Vector model

Vector model of documents: Let $a_1, \ldots, a_n$ be terms, $D_1, \ldots, D_m$ documents, and ***relevance matrix*** $W = (w_{ij})$ of type m, n,

$$w_{ij} \in \langle 0, 1 \rangle \begin{cases} 0 & \text{is irrelevant} \\ 1 & \text{is relevant} \end{cases}$$

Query $Q = (q_1, \ldots, q_n)$

- $S(Q, D_i) = \sum_i q_i w_{ij}$ ***similarity coefficient***
- $head(sort(S(Q, D_i)))$ answer

Vector model: pros & cons

CONS: *doesn't take into account ?"and"? ?"or"?*

PROS: *possible improvement:*

- normalization of weights
  - ***Term frequency*** *TF*
  - ***Inverted document frequency*** $IDF \equiv \log_2 \frac{m}{k}$
  - Distinction of terms
- normalization of weights for document: $\frac{TD}{\sqrt{\sum_j TD_j^3}}$
- normalization of weights for query: $\left(\frac{1}{2} \times \frac{\frac{1}{2}TF}{\max TF_i}\right) \times \log_2 \frac{m}{k}$

[POK, *pages 85–113*].

Automatic structuring of texts

- Interrelations *between documents* in TIS.
- Encyclopedia (OSN, Funk *and* Wagnalls New Encyclopedia).

- [SBA]
  `http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs`
- *Google/CiteSeer*: „*automatic structuring of text files*“

Similarity of documents

- Most often cosine measure – advantages.
- Detailed overview of similarity functions see chapter 5.7 from [KOR] (similarity).

Lexicon storage

[MeM] Mehryar Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing, *Natural Language Engineering*, 2(1):61–80, 1996.
`http://www.research.att.com/~mohri/cl1.ps.gz`

Signature methods

Search methods IV. Text preprocessing of text and pattern (signature methods).

- chained
- layered

More [POK, pages 65–76], see [MAR].