PV030 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2012

Part I

Information about the course PV030

Introduction

- Petr Sojka, `sojka@fi.muni.cz`
- Consulting hours Spring 2012:
  Wednesday 13:00–13:45
  Friday 10:00–11:50
  or write an email with other suggestions to meet.
- Room C523/522, fifth floor of block C, Botanická 68a.
- Course homepage: `http://www.fi.muni.cz/~sojka/PV030/`
- Seminar (Thu 12:00–12:50, C511 → B311).

Topics and classification of the course

Prerequisites:
It is expected the student having basic knowledge of the theory of automata and formal languages (IB005), elementary knowledge of theories of complexity, software programming and systems.
Classification:
There is a system of classification based on written mid-term (30 points) and final (70 points) exams. In addition, one can get additional premium points based on the activities during lectures.
Classification scale (changes reserved) z/k/E/D/C/B/A correspond to obtaining 48/54/60/66/72/78/84 points.
Dates of [final] exams will be announced via IS.muni.cz (probably three terms).

Topics

My books focus on timeless truth.
D. E. Knuth, Brno, 1996

An emphasis will be given to the explanation of basic principles, algorithms and (software) design techniques, creation and implementation of textual information systems (TIS)—storage and information retrieval.

Syllabus

① Basic notions. TIS (text information system). Classification of information systems. From texts to Watson.

② Searching in TIS. Searching and pattern matching classification and data structures. Algorithms of Knuth-Morris-Pratt, Aho-Corasick, reg. expr.

③ Algorithms of Boyer-Moore, Commentz-Walter, Buczilowski.

④ Theory of automata for searching. Classification of searching problems. Searching with errors.

⑤ Indexes. Indexing methods. Data structures for searching and indexing.

⑥ Google as an example of search and indexing engine. Pagerank. Signature methods.

⑦ Query languages and document models: Boolean, vector, probabilistic, MMM, Paice.

Syllabus (cont.)

⑧ Data compression. Basic notions. Entropy.

⑨ Statistical methods.

⑩ Compression methods based on dictionary.

❶ Syntactic methods. Context modeling. Language modeling. Corpora linguistics.

❷ Spell checking. Filtering information channels. Document classification. Neural nets for text compression.

Textbooks

- [MEL] Melichar, B.: *Textové informační systémy*, skripta ČVUT Praha, 2. vydání, 1996.
- [POK] Pokorný, J., Snášel, V., Húsek D.: *Dokumentografické informační systémy*, Karolinum Praha, 1998.
- [KOR] Korfhage, R. R.: *Information Storage and Retrieval*, Wiley Computer Publishing, 1997.
- [SMY] Smyth, B.: *Computing Patterns in Strings*, Addison Wesley, 2003.
- [KNU] Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, Sorting and Searching, Second edition, 1998.
- [WMB] Witten I. H., Moffat A., Bell T. C.: *Managing Gigabytes: compressing and indexing documents and images*, Second edition, Morgan Kaufmann Publishers, 1998.

Other study materials

- [HEL] Held, G.: *Data and Image Compression*, Tools and Techniques, John Wiley & Sons, 4. vydání 1996.
- [MEH] Melichar B., Holub J., A 6D Classification of Pattern Matching Problems, Proceedings of The Prague Stringology Club Workshop '97, Prague, July 7, CZ.
- [G00] Brin S., Page, L.: The anatomy of a Large-Scale Hypertextual Web Search Engine. WWW7/Computer Networks 30(1–7): 107–117 (1998). `http://dbpubs.stanford.edu:8090/pub/1998-8`
- [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`

Other study materials (cont.)

- [Sch] Schmidhuber J.: Sequential neural text compression, *IEEE Transactions on Neural Networks* 7(1), 142–146, 1996, `http://www.idsia.ch/~juergen/onlinepub.html`
- [SBA] Salton G., Buckley Ch., Allan J.: Automatic structuring of text files, *Electronic Publishing* 5(1), p. 1–17 (March 1992). `http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs.htm`
- [WWW] *web pages of the course* `~sojka/PV030/`, DIS seminars `http://www.inf.upol.cz/dis`, `http://nlp.fi.muni.cz/`, The Prague Stringology Club Workshop 1996–2008 `http://cs.felk.cvut.cz/psc/`
- Jones, S. K., Willett: *Readings in Information Retrieval*, Morgan Kaufman Publishers, 1997.

Other study materials (cont.)

- Bell, T. C., Cleary, J. G., Witten, I. H.: *Text Compression*, Prentice Hall, Englewood Cliffs, N. J., 1991.
- Storer, J.: *Data Compression: Methods and Theory*, Computer Science Press, Rockwille, 1988.
- journals *ACM Transactions on Information Systems*, *Theoretical Computer Science*, *Neural Network World*, *ACM Transactions on Computer Systems*, *Knowledge Acquisition*.

`knihovna.muni.cz`, `umarecka.cz` (textbook Pokorný),

Part II

Basic notions of TIS

TIS—motivation

$$\begin{tikzcd}
\text{reality} \arrow[r, leftrightarrow] & \text{data} \\
\text{information need} \arrow[u] \arrow[r, leftrightarrow] & \text{query} \arrow[u]
\end{tikzcd}$$

- ☞ Abstractions and mappings in information systems.
- ☞ Information needs about the reality—queries above data.
- ☞ Jeopardy game: Watson.

Notions of (T)IS

Definition: ***Information system*** is a system that allows purposeful arrangement of collection, storage, processing and delivering of information.

Definition: ***Ectosystem*** consists of IS users, investor of IS, and entrepreneur (user, funder, server). In the example of is.muni.cz they are users of IS, MU represented by bursar, and ICS and IS teams. Ectosystem is not under control of IS designer.

Definition: ***Endosystem*** consists of hardware used (media, devices), and software (algorithms, data structures) and is under control of IS designer.

Demands on TIS

- ☞ effectiveness (user)
- ☞ economics (funder)
- ☞ efficiency (server)

and from different preferences implied compromises. Our view will be view of TIS architect respecting requests of IS ectosystem. For topics related to ectosystem of IS see PV045 Management IS.

From data to wisdom

- Data: concrete representation of a message in a form of sequence of symbols of an alphabet.
- Information: reflection of the known or the expected substance of realities. An information depends on the intended subject. Viewpoints:
  - quantitative (information theory);
  - qualitative (meaning, semantics);
  - pragmatical (valuation: significance, usefulness, usability, periodicity, up-to-dateness, credibility;
  - the others (promptness, particularity, completeness, univocality, availability, costs of obtaining).
- Knowledge (znalost).
- Wisdom (moudrost).

Information process

Definition: ***Information process*** is a process of formation of information, its representation in a form of data, its processing, providing, and use. Operations with information correspond to this process.

Data/signals $\longrightarrow$ Information $\longrightarrow$ Knowledge $\longrightarrow$ Wisdom.

IS classification by the prevailing function

① Information retrieval systems.

② Database management systems (DBMS), relational DB (PB154, PB155, PV003, PV055, PV136, PB114).

③ Management information systems (PV045).

④ Decision support systems (PV098).

⑤ Expert systems, question answering systems, knowledge-based systems (PA031).

⑥ Information service systems (web 2.0).

IS classification by the prevailing function (cont.)

6. Specific information systems (geographical PV019, PA049, PA050, medical PV048, environmental PV044, corporate PV043, state administration PV058, PV059, librarian PV070); and also PV063. Application of database systems.

Related fields taught in FI:
Software engineering (PA102, PA105).
Similarity searching in multimedia data (PA128).
Efficient use of database systems (PA152).
Introduction to information retrieval (PV211).

Diversity of TIS perspectives

↖ ↗

Information retrieval system — Expert system

DBMS

Database system — Management system

↙ ↘

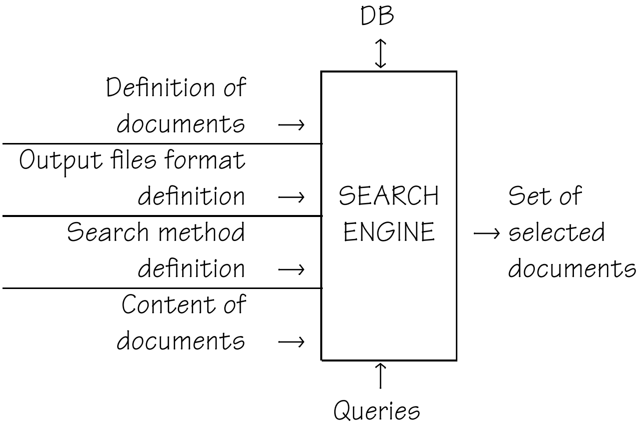
Mini questionnaire

1. What do you expect from this course? What was your motivation to enroll? Is the planned syllabus fine? Any changes or surprises?
2. What do you not expect (you would rather eliminate)?
3. Which related courses have you already passed?
4. Practising IS usage (as a user)
   a) Which (T)IS do you use?
   b) Intensity? Frequency? How many searching per month?
   c) Are you satisfied with it?
5. IS creation (server)
   a) Which (T)IS and its component have you realized? Area, size?
   b) Are you satisfied with it? Bottlenecks?

Information retrieval systems (IRS)—principles

$$\boxed{\begin{array}{ccccc} & & \text{DB} & & \\ & & \updownarrow & & \\ \text{Query} & \longrightarrow & \begin{array}{c}\text{Search}\\ \text{engine}\end{array} & \longrightarrow & \begin{array}{l}\text{Set of selected}\\ \text{documents}\end{array} \end{array}}$$

An empty IRS


DB
Definition of documents
Output files format definition
Search method definition
Content of documents
SEARCH ENGINE
Set of selected documents
Queries

Searching—formalization of the problem

Concatenation: string of beads. A bead $\longrightarrow$ an element. Indexing of elements by natural numbers. Not necessarily numbers, but labels.

0) Every element has unique label.
1) Every labeled element x (except for the leftmost one) has a clear predecessor referred to as $pred(x)$.
2) Every labeled element x (except for the rightmost one) has a clear successor referred to as $succ(x)$.
3) If the element x is not the leftmost one, $x = succ(pred(x))$.
4) If the element x is not the rightmost one, $x = pred(succ(x))$.
5) For every two different elements x and y, there exists a positive number k that is either $x = succ^k(y)$ or $x = pred^k(y)$.

Searching—formalization of the problem (cont.)

The concatenation term:

Definition: ***a string*** is a set of elements which meets the rules 0)–5).

Definition: ***a linear string***: a string that has a finitely many elements including the leftmost and rightmost ones.

Definition: ***a necklace***.

Definition: ***an alphabet A***. ***Letters of the alphabet***. A^+. ***An empty string*** ε.

Definition: ***a finite chain*** $A^* = A^+ \cup \{\varepsilon\}$.

Definition: ***a linear string over A***: a member of A^+.

Definition: ***a pattern***. ***A text***.

IRS—classification

① Classification according to the passing direction: left-to-right/right-to-left.

② Classification according to (pre)processing of the text and the pattern:

- ad fontes (searching in the text itself);
- text surrogate (searching in the substitution of the text);
- substitutions:
  an index: an ordered list of significant elements together with references to the original text;
  a signature: a string of indicators that shows the occurrence of significant elements in the text.

IRS—classification (cont.)

		text preprocessing	
		no	yes
pattern preprocessing	no	I	III
	yes	II	IV

- I – elementary algorithms
- II – creating a search engine
- III – indexing methods
- IV – signature methods

Searching—the formulation of the problem

Classification according to the cardinality of the patterns' set:

① Search for a single pattern V in the text T. The result: yes/no.

② Search for a finite set of patterns $P = \{v_1, v_2, \ldots, v_k\}$. The result: information about position of some of the entered patterns.

③ Search for an infinite set of patterns assigned by a regular expression R. R defines a potentially infinite set $L(R)$. The result: information about position of some of the patterns from $L(R)$.

Alternatives to the formulation of the searching problem:

a) the first occurrence;

b) the all occurrences without overlapping;

c) the all occurrences including overlapping.

Part III

Exact search

Naïve search, brute force search, rudimentary search algorithm

```
proc Brute-Force-Matcher(PATTERN,TEXT):
T:=length[TEXT]; P:=length[PATTERN];
for i:=0 to T-P do
  if PATTERN[1..P]=TEXT[i+1..i+P]
  then print "The pattern was found at the position i.";
```

Time complexity analysis of naïve search

- The complexity is measured by number of comparison, the length of a pattern P, the length of text T.
- The upper estimate $S = P \cdot (T - P + 1)$, thus $O(P \times T)$.
- The worst case PATTERN $= a^{P-1}b$, TEXT $= a^{T-1}b$.
- Natural languages: (average) complexity (number of comparison) substantially smaller, since the equality of prefixes doesn't occur very often. For English: $S = C_E \cdot (T - P + 1)$, C_E empirically measured 1.07, i. e. practically linear.
- C_{CZ}? C_{CZ} vs. C_E?
- Any speedups? An application of several patterns? An infinite number?
- We will see the version (S, Q, Q') of the algorithm in the seminar.

Naïve search—algorithms

Express the time complexity of the following search algorithms using the variables c and s, where c is the number of the tests and these statements are true:

- if the index i is found, then $c = i$ and $s = 1$;
- otherwise, $c = T$ and $s = 0$.

Naïve search—algorithm S

```
     input:  var TEXT : array[1..T] of word;
                 PATTERN : word;
     output (in the variable FOUND): yes/no
1    I:=1;
c    while I≤ T do
     begin
c        if TEXT[I]=PATTERN then break;
c-s      inc(I);
     end;
2    FOUND:=(I≤T);
```

On the left side, there is the time complexity of the statements.
And so the overall time complexity is $O(T) = 3c - s + 3$.
The maximum complexity (which is commonly stated) is $O(T) = 3T + 3$.

Algorithm Q or how about using the end stop/skid (zarážka)

```
      input:  var TEXT : array[1..T+1] of word; PATTERN : word;
      output (in the variable FOUND): yes/no
1     I:=1;
1     TEXT[T+1]:=PATTERN;
c     while TEXT[I]<>PATTERN do
c-1      inc(I);
2     FOUND:=(I<>T+1)
```

In this case, the index is always found; therefore it is stated on the last but one line of the algorithm that the complexity is $c - 1$ instead of $c - s$ (although they are equivalent). Furthermore, it is necessary to realize that the maximal possible value of c is greater by one than in the previous algorithm (stating $c + 1$ instead of c would not be correct, though). The overall complexity: $O(T) = 2c + 3$. The maximum complexity: $O(T) = 2T + 5$.

Algorithm Q' or how about using the cycle expansion

```
                 input: var TEXT : array[1..T+1] of word;
                            PATTERN : word;
                 output (in the variable FOUND): yes/no
1                I:=1;
1                TEXT[T+1]:=PATTERN;
$\lceil c/2 \rceil$          while TEXT[I]<>PATTERN do
                   begin
$\lfloor c/2 \rfloor$              if TEXT[I+1]=PATTERN then break;
$\lfloor (c-1)/2 \rfloor$          I:=I+2;
                   end;
3                FOUND:=(I<T)or(TEXT[T]=PATTERN);
```

The overall complexity: $O(T) = c + \lfloor (c-1)/2 \rfloor + 5$.

The maximum complexity: $O(T) = T + \lfloor T/2 \rfloor + 6$.

The condition at the end of the algorithm guarantees its functionality (however, it is not the only way of handling the cycle incrementation by two).

Outline (week two)

1. Watson
2. Exact search methods I (without pattern preprocessing) – completion.
3. Exact search methods II (with pattern preprocessing, left to right): KMP (animation), Rabin-Karp, AC.
4. Search with an automaton.

Evaluation of questionnaire

① Yes: syllabus suits expectations; positively is awaited dissect of Google; indexing and search; examples.

② No: too much theory, deep digestion of algorithms.

③ Examples.

④ This year: further enrichment of information retrieval part (Google), textual (mathematical) digital libraries and languages enhancements of TIS (on the example of Watson).

Motivation

① Search in text editor (Vim, Emacs), in the source code of a web page.

② Data search (biological molecules approximated as sequences of nucleotides or amino acids).

③ Literature/abstracts search—recherche, corpus linguistics.

The size of available data doubles every 18 months (Moore's law) $\longrightarrow$ higher effectiveness of algorithms needed.

Left-to-right direct search methods

During the preprocessing, structure of the query pattern(s) is examined and, on that basis, the search engine is built (on-the-fly).

Definition: ***exact*** (vs. ***fuzzy (proximitní)***) search aims at exact match (localization of searched pattern(s)).

Definition: ***left-to-right (LR, sousměrné)*** (vs. ***right-to-left (RL, protisměrné)***) search compares query pattern to the text from left to right (vs. right to left).

Left-to-right methods

① 1 query pattern (vzorek):
- Shift-Or algorithm.
- Karp-Rabin algorithm, (KR, 1987).
- Knuth-Morris-Pratt algorithm, (KMP, *designed* (MP) in 1970, *published* 1977).

② n patterns: Aho-Corasick algorithm, (AC, 1975).

③ ∞ patterns: *construction of a search engine* (finite automaton) *for the search of a potentially* infinite set *of* patterns (*given as* regular *expression*).

Shift-Or algorithm

- ☞ Pattern $v_1 v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- ☞ Incidence matrix X $(m \times c)$, $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- ☞ Let matrix column X corresponding to a_j is named A_j.
- ☞ At the beginning, we put unitary vector/column into R. In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction: $R := \text{SHIFT}(R)\ \text{OR}\ A_j$.
- ☞ Algorithm stops successfully when 0 appears at the bottom-most position in R.

Shift-Or algorithm (cont.) – *example*

Example: $V = vzorek$ over $\Sigma = \{e, k, o, r, v, z\}$.
Cf. [POK, page 31–32].

Karp-Rabin search

Quite different approach: usage of hash function. Instead of matching of pattern with text on every position, we check the match only when pattern 'looks similar' as searched text substring. For similarity, a hash function is used. It has to be

- efficiently computable,
- and it should be good at separating different strings (close to perfect hashing).

KR search is quadratic at the worst case, but on average $O(T + V)$.

Karp-Rabin search (cont.)—implementation

```
#define REHASH(a, b, h) (((h-a*d)<<1+b)
void KR(char *y, char *x, int n, int m) {
int hy, hx, d, i;
/* preprocessing: computation of $d = 2^{m-1}$ */
d=1; for (i=1; i<m; i++) d<<=1;
hx=hy=0;
for (i=0; i<m; i++)
  { hx=((hx<<1)+x[i]); hy=((hy<<1)+y[i]); }
/* search */
for (i=m; i<=n; i++) {
  if (hy==hx) && strncmp(y+i-m,x,m)==0) OUTPUT(i-m);
  hy=REHASH(y[i-m], y[i], hy);
} }
```

Karp-Rabin search (cont.)—*example*

Example: ([HCS, Ch. 6]) $V = ing$, $T = string\ matching$.
Preprocessing: $hash = 105 \times 2^2 + 110 \times 2 + 103 = 743$.
Search:

T=	s	t	r	**i**	**n**	**g**	
hash=			806	797	776	**743**	678

m	a	t	c	h	**i**	**n**	**g**
585	443	746	719	766	709	736	**743**

Part IV

Exact search of one pattern

Morris-Pratt algorithm (MP)

Idea: Inefficiency of naïve search are caused by the fact that in the case of mismatch the pattern is shifted by only one position to the right and checking starts from the beginning. This does not use the information that was gained by the inspection of text position that failed. The idea is to shift as much as possible so that we do not have to go back in searched text.

The main part of the (K)MP algorithm

```
**var** $text$: array[1..$T$] of char; $pattern$: array[1..$V$] of char;
$i$, $j$: integer; $found$: boolean;
$i := 1$;                                              ▷ text index
$j := 1$;                                              ▷ pattern index
**while** $(i \leq T)$ and $(j \leq V)$ **do**
    **while** $(j > 0)$ and $(text[i] \neq pattern[j])$ **do**
        $j := h[j]$;
    **end while**
    $i := i + 1$; $j := j + 1$
**end while**
$found := j > V$;                  ▷ if found, it is on the position $i - V$
```

Analysis of (K)MP

- ☞ $O(T)$ complexity plus complexity of preprocessing (creation of the array h).
- ☞ Animation of tracing of the main part of KMP.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1 v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1} t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j? How to compute h?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1 v_2 \dots v_{k-1}$ is suffix of $v_1 v_2 \dots v_{j-1}$, e.g. $v_1 v_2 \dots v_{k-1} = v_{j-k+1} v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = 0$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ $(v_1 v_2 \dots v_j = t_{i-j+1} t_{i-j+2} \dots t_{i-1} t_i)$.
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Construction of *h* for KMP

```
i:=1; j:=0; h[1]:=0;
while (i<V) do
  begin while (j>0) and (v[i]<>v[j]) do j:=h[j];
    i:=i+1; j:=j+1;
    if (i<=V) and (v[i]=v[j])
    then h[i]:=h[j] else h[i]:=j (*MP*)
  end;
```

Complexity of *h* computation, *e.g.* preprocessing, is $O(V)$, thus in total $O(T+V)$.
Example: *h* for *ababa*. KMP vs. MP.

Universal search algorithm,

that uses transition table g derived from the searched pattern, (g relates to the transition function δ of FA):

```
var i,T:integer; found: boolean;
text: array[1..T] of char; state,q0: TSTATE;
g:array[1..maxstate,1..maxsymb] of TSTATE;
F: set of TSTATE;...
begin
  found:= FALSE; state:= q0; i:=0;
  while (i <= T) and not found do
    begin
      i:=i+1; state:= g[state,text[i]];
      found:= state in F;
    end;
  end;
```

How to transform pattern into g?

Search engine (SE) for left-to-right search

☞ ***SE for left-to-right search*** $A = (Q, T, g, h, q_0, F)$

- Q is a finite set of states.
- T is a finite input alphabet.
- g: $Q \times T \longrightarrow Q \cup \{\text{fail}\}$ is a forward state-transition function.
- h: $(Q - q_0) \longrightarrow Q$ is a backward state-transition function.
- q_0 is an initial state.
- F is a set of final states.

☞ ***A depth of the state*** q: $d(q) \in N_0$ is a length of the shortest forward sequence of the state transitions from q_0 to q.

Search engine (cont.)

☞ Characteristics g, h:

- $g(q_0, a) \neq \underset{\sim\sim\sim}{\text{fail}}$ for $\forall a \in T$ (there is no backward transition in the initial state).
- If $h(q) = p$, then $d(p) < d(q)$ (the number of the backward transitions is restricted from the top by a multiple of the maximum depth of the state c and the sum of the forward transitions V). So the speed of searching is linear in relation to V.

SE configuration, transition

- ☞ ***SE configuration*** (q, w), $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ ***An initial configuration of SE*** (q_0, w), w is the entire searched text.
- ☞ ***An accepting configuration of SE*** (q, w), $q \in F$, w is the not yet searched text, the found pattern is immediately before w.
- ☞ ***SE transition***: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
    - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ ***forward transition*** for $\forall w \in T^*$.
    - $h(q) = p$, then $(q, w) \vdash (p, w)$ ***backward transition*** for $\forall w \in T^*$.

Searching with SE

During the forward transition, a single input symbol is read and the engine switches to the next state p. However, if $g(q, a) = \text{fail}$, the backward transition is executed without reading an input symbol. $S = O(T)$ (we measure the number of SE transitions).

Construction of the KMP SE for pattern $v_1 v_2 \dots v_V$

① An initial state q_0.

② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1 v_2 \dots v_j v_{j+1}$.

③ For q_0, we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.

④ $g(q, a) =$ fail for $\forall q$ and a, for which $g(q, a)$ has not been defined in the previous steps.

⑤ A state that corresponds to the complete pattern is the final one.

⑥ The backward state-transition function h is defined on the page 51 by the below mentioned algorithm.

Outline (week two)

1. Summary of the previous lecture, searching with SE.
2. Left-to-right search of n patterns algorithms. (AC, NFA $\longrightarrow$ DFA.)
3. Left-to-right search of infinite patterns algorithms.
4. Regular expressions (RE).
5. Direct construction of (N)FA for given RE.

Part V

Search of a finite set of patterns

Search of a set of patterns

SE for left-to-right search of a set of patterns $p = \{v^1, v^2, \ldots, v^P\}$.

Instead of repeated search of text for every pattern, there is only "one" pass (FA).

Common SE algorithm

```
var text: array[1..T] of char;
  i: integer; found: boolean; state: tstate;
  g: array[1..maxstate,1..maxsymbol] of tstate;
  h: array[1..maxstate] of tstate; F: set of tstate;
found:=false; state:=q0; i:=0;
while (i<=T) and not found do
begin i:=i+1;
  while g[state,text[i]]=fail do state:=h[state];
  state:=g[state,text[i]]; found:=state in F
end
```

Common SE algorithm (cont.)

- Construction of the state-transition functions h, g?
- How about for P patterns? The main idea?
- Aho, Corasick, 1975 (AC search engine).

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

1. An initial state q_0.
2. $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i, for $\forall i \in \{1, \dots, P\}$.
3. For q_0, we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
4. $g(q, a) =$ fail for $\forall q$ and a, for which $g(q, a)$ has not been defined in the previous steps.
5. A state that corresponds to the complete pattern is the final one.

An *example*: p ={he, she, her} *over* T ={h, *e*, r, s, x}, where x is anything else than {h, *e*, r, s}.

The failure function h (AC II)

Construction of h for AC SE for a set of patterns $p = \{v^1, v^2, \ldots, v^p\}$

At first, we define the failure function f inductively relative to the depth of the states this way:

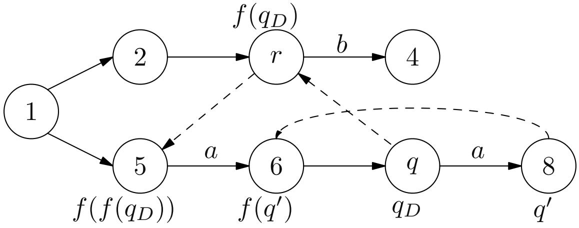
1. For $\forall q$ of the depth 1, $f(q) = q_0$.
2. Let us assume that f is defined for each state of the depth d and lesser. The variable q_D denotes the state of the depth d and $g(q_D, a) = q'$. Then we compute $f(q')$ as follows:

```
$q := f(q_D)$;
while $g(q, a) = \text{fail}$ do $q := f(q)$;
$f(q') := g(q, a)$.
```

The failure function h (AC II, cont.)

- The cycle terminates, since $g(q_0, a) \neq \utilde{\text{fail}}$.
- If the states q, r represent prefixes u, v of some of the patterns from p, then $f(q) = r \Leftrightarrow v$ is the longest proper suffix u.

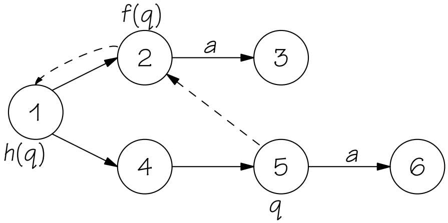
The failure function *h* (AC III)



Construction of h for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^p\}$ (cont.)

- We could use f as the backward state-transition function h, however, redundant backward transitions would be performed.
- We define function h inductively relative to the depth of the states this way:
    - For $\forall$ state q of the depth 1, $h(q) = q_0$.
    - Let us assume that h is defined for each state of the depth d and lesser. Let the depth q be $d + 1$. If the set of letters, for which is in a state $f(q)$ the value of the function g different from fail, is the subset of the set of letters, for which is the value of the function g in a state q different from fail, then $h(q) := h(f(q))$, otherwise $h(q) := f(q)$.

Construction of h for AC SE (cont.)



Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

① K is a finite set of inner states.

② T is a finite input alphabet.

③ δ is a projection from $K \times T$ to K.

④ $q_0 \in K$ is an initial state.

⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

① ***Completely specified automaton*** if δ is defined for every pair $(q, a) \in K \times T$, otherwise ***incompletely specified automaton***.

② ***Configuration M*** is a pair (q, w), where $q \in K$, $w \in T^*$ is the not yet searched part of the text.

③ ***An initial configuration M*** is (q_0, w), where w is the entire text to be searched.

④ ***An accepting configuration M*** is (q, w), where $q \in F$ and $w \in T^*$.

Searching with FA M

During the transition, a single input symbol is read and the engine switches to the next state p.

- ☞ ***Transition M***: is defined by a state and an input symbol; relation $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; if $\delta(q, a) = p$, then $(q, aw) \vdash (p, w)$ for every $\forall w \in T^*$.
- ☞ ***The kth power***, ***transitive*** or more precisely ***transitive reflexive closure*** of the relation $\vdash$: $\vdash^k$, $\vdash^+$, $\vdash^*$.
- ☞ $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w') \text{ for some } q \in F, w' \in T^*\}$ ***the language accepted by FA*** M.
- ☞ time complexity $O(T)$ (we measure the number of transitions of FA M).

Nondeterministic FA

Definition: ***Nondeterministic finite automaton*** (NFA) is $M = (K, T, \delta, q_0, F)$, where K, T, q_0, F are the same as those in the deterministic version of FA, but $\delta : K \times T \longrightarrow 2^K$ $\delta(q, a)$ is now ***a set*** of states.

Definition: $\vdash \in (K \times T^*) \times (K \times T^*)$ ***transition***: if $p \in \delta(q, a)$, then $(q, aw) \vdash (p, w)$ for $\forall w \in T^*$.

Definition: a final state, $L(M)$ analogically as in DFA.

Construction of SE (DFA) from NFA

Theorem: for *every* nondeterministic finite automaton $M{=}(K,T,\delta,q_0,F)$, *we can build* <u>deterministic</u> finite automaton $M'{=}(K',T,\delta',q'_0, F')$ such that $L(M) = L(M')$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):
Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.
Output: deterministic FA.

① $K'=\{\{q_0\}\}$, state $\{q_0\}$ in unmarked.

② If there are in K' all the states marked, continue to the step 4.

③ We choose from K' unmarked state q':
- $\delta'(q', a) = \bigcup\{\delta(p, a)\}$ for,$\forall p \in q'$ and $a \in T$;
- $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
- we mark q' and continue to the step 2.

④ $q_0' = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of g for SE

Construction g' for SE for a set of patterns $p = \{v^1, v^2, \ldots, v^P\}$

1. We create NFA M:
    - An initial state q_0.
    - For $\forall a \in T$, we define $g(q_0, a) = q_0$.
    - For $\forall i \in \{1, \ldots, P\}$, we define $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \ldots b_{j+1}$ of the pattern v^i.
    - The state corresponding to the entire pattern is the final one.
2. …and its corresponding DFA M' with g'.

Part VI

Search for an infinite set of patterns

Regular expression (RE)

Definition: ***Regular expression E over the alphabet A***:

1. $\varepsilon, \mathbf{0}$ are RE and for $\forall a \in A$ is a RE.
2. If x, y are RE over A, then:
   - $(x + y)$ is RE (union);
   - $(x.y)$ is RE (concatenation);
   - $(x)^*$ is RE (iteration).

A convention about priority of regular operations:
union $<$ concatenation $<$ iteration.
Definition: Thereafter, we consider as a ***(generalized) regular expression*** even those terms that do not contain, with regard to this convention, the unnecessary parentheses.

Value of RE

1. $h(\mathbf{O}) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$
2. - $h(x + y) = h(x) \cup h(y)$
   - $h(x.y) = h(x).h(y)$
   - $h(x^*) = (h(x))^*$

☞ $h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$

☞ The value of RE is a regular language (RL).

☞ Every RL can be represented as RE.

☞ For $\forall$ RE V $\exists$ FA M: $h(V) = L(M)$.

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $\mathbf{0}.x = \mathbf{0}$ inverse element for concatenation

A9: $x + \mathbf{0} = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Outline (week four)

① Summary of the previous lecture.

② Regular expressions, value of RE, characteristics.

③ Derivation of regular expressions.

④ Direct construction of equivalent DFA for given RE by derivation.

⑤ Derivation of regular expressions by position vector.

⑥ Right-to-left search (BMH, CW, BUC).

Similarity of regular expressions

Theorem: the axiomatization of RE is complete and consistent.

Definition: regular expressions are termed as ***similar***, when they can be mutually conversed using axioms A1 to A11.

Theorem: similar regular expressions have the same value.

Length of a regular expression

Definition: ***the length d(E) of the regular expression E***:

① If E consists of one symbol, then $d(E) = 1$.

② $d(V_1 + V_2) = d(V_1) + d(V_2) + 1$.

③ $d(V_1.V_2) = d(V_1) + d(V_2) + 1$.

④ $d(V^*) = d(V) + 1$.

⑤ $d((V)) = d(V) + 2$.

Note: the length corresponds to the syntax of a regular expression.

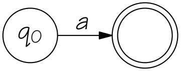
Construction of NFA for given RE

Definition: ***a generalized NFA*** allows ε-transitions (transitions without reading of an input symbol).

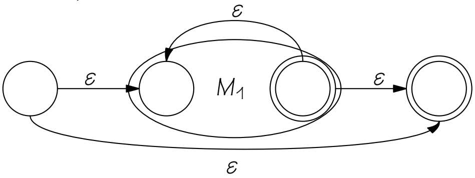
Theorem: for every RE E, we can create FA M such that $h(E) = L(M)$.
Proof: by structural induction relative to the RE E:

Construction of NFA for given RE (a proof)

① $E = a$

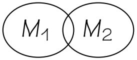


② $E = E_1^*$ M_1 automaton for E_1 $(h(E_1) = L(M_1))$

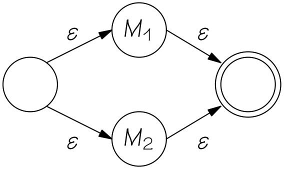


Construction of NFA for given RE (cont. of a proof)

③ $E = E_1 \cdot E_2$



④ $E = E_1 + E_2$ M_1, M_2 automata for E_1, E_2 ($h(E_1) = L(M_1)$,



$h(E_2) = L(M_2)$)

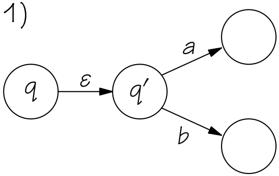
Construction of NFA for given RE (cont.)

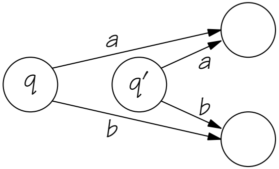
- ☞ No more than two *edges come* out of every state.
- ☞ No *edges come* out of the final states.
- ☞ The number of the states $M \leq 2 \cdot d(E)$.
- ☞ The simulation of automaton M is performed in $O(d(E)T)$ time and in $O(d(E))$ space.

NFA simulation

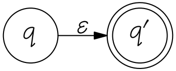
For the following methods of NFA simulation, we must remove the ε-transitions. We can achieve it with the well-known procedure:

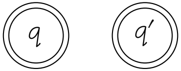
1)





2)





NFA simulation (cont.)

We represent a state with a Boolean vector and we pass through all the paths at the same time. There are two approaches:

☞ The general algorithm that use a transition table.

☞ Implementation of the automaton in a form of (generated) program for the particular automaton.

Direct construction of (N)FA for given RE

Let E is a RE over the alphabet T. Then we create FA $M = (K, T, \delta, q_0, F)$ such that $h(E) = L(M)$ this way:

① We assign different natural numbers to all the occurrences of the symbols of T in the expression E. We get E'.

② A set of starting symbols $Z = \{x_i :$ a string of $h(E')$ can start with the symbol x_i, $x_i \neq \varepsilon\}$.

③ A set of neighbours $P = \{x_i y_j :$ symbols $x_i \neq \varepsilon \neq y_j$ can be next to each other in a string of $h(E')\}$.

④ A set of ending symbols $F = \{x_i :$ a string of $h(E')$ can end with the symbol $x_i \neq \varepsilon\}$.

⑤ A set of states $K = \{q_0\} \cup Z \cup \{y_j : x_i y_j \in P\}$.

⑥ A transition function δ:

- $\delta(q_0, x)$ contains x_i for,$\forall x_i \in Z$ that originate from numbering of x.
- $\delta(x_i, y)$ contains y_j for,$\forall x_i y_j \in P$ such that y_j originates from numbering of y.

⑦ F is a set of final states, a state that corresponds to E is q_0.

Direct construction of (N)FA for given RE (cont.)

Example 1: $R = ab^*a + ac + b^*ab^*$.

Example 2: $R = ab^* + ac + b^*a$.

Derivation of a regular expression

Definition: ***derivation $\frac{dE}{dx}$ of the regular expression E by a string $x \in T^*$***:

1. $\frac{dE}{d\varepsilon} = E$.
2. For $a \in T$, these statements are true:

$$
\begin{aligned}
\frac{d\varepsilon}{da} &= \mathbf{0} \\
\frac{db}{da} &= \begin{cases} \mathbf{0} & \text{if } a \neq b \\ \varepsilon & \text{if } a = b \end{cases} \\
\frac{d(E+F)}{da} &= \frac{dE}{da} + \frac{dF}{da} \\
\frac{d(E.F)}{da} &= \begin{cases} \frac{dE}{da} \cdot F + \frac{dF}{da} & \text{if } \varepsilon \in h(E) \\ \frac{dE}{da} \cdot F & \text{otherwise} \end{cases} \\
\frac{d(E^*)}{da} &= \frac{dE}{da} \cdot E^*
\end{aligned}
$$

Derivation of a regular expression (cont.)

③ For $x = a_1 a_2 \ldots a_n$, $a_i \in T$, these statements are true

$$\frac{dE}{dx} = \frac{d}{da_n}\left(\frac{d}{da_{n-1}}\left(\cdots \frac{d}{da_2}\left(\frac{dE}{da_1}\right)\cdots\right)\right).$$

Characteristics of regular expressions

Example: Derive $E = fi + fi^* + f^*ifi$ by i and f.
Example: Derive $(o^*sle)^*cno$ by o, s, l, c and $osle$.

Theorem: $h\left(\frac{dE}{dx}\right) = \{y : xy \in h(E)\}$.
Example: Prove the above-mentioned statement. Instruction: use structural induction relative to E and x.

Definition: ***Regular expressions x, y are similar*** if one of them can be transformed to the other one with axioms of the axiomatic theory of RE (Salomaa).

Example: Is there a RE similar to $E = fi + fi^* + f^*ifi$ that has length 7, 15?

Direct construction of DFA for given RE (by RE derivation)

Brzozowski (1964, Journal of the ACM)
Input: RE E over T.
Output: FA $M = (K, T, \delta, q_0, F)$ such that $h(E) = L(M)$.

1. Let us state $Q = \{E\}$, $Q_0 = \{E\}$, $i := 1$.
2. Let us create the derivation of all the expressions of Q_{i-1} by all the symbols of T. Into Q_i, we insert all the expressions created by the derivation of the expressions of Q_{i-1} that are not similar to the expressions of Q.
3. If $Q_i \neq \emptyset$, we insert Q_i into Q, set $i := i + 1$ a move to the step 2.
4. For $\forall \frac{dF}{dx} \in Q$ and $a \in T$, we set $\delta\left(\frac{dF}{dx}, a\right) = \frac{dF}{dx'}$, in case that the expression $\frac{dF}{dx'}$ is similar to the expression $\frac{dF}{d\,xa}$. (Concurrently $\frac{dF}{dx'} \in Q$.)
5. The set $F = \left\{\frac{dF}{dx} \in Q : \varepsilon \in h\left(\frac{dF}{dx}\right)\right\}$.

Example: RE= $R = (0+1)^*1$.
$Q = Q_0 = \{(0+1)^*1\}, i = 1$
$Q_1 = \{\frac{dR}{d0} = R, \frac{dR}{1}\} = \{(0+1)^*1 + \varepsilon\}$
$Q_2 = \{\frac{(0+1)^*1+\varepsilon}{d0} = R, \frac{(0+1)^*1+\varepsilon}{d1} = (0+1)^*1 + \varepsilon\} = \emptyset$

Example: RE= $(10)^*(00)^*1$.

For more, see Watson, B. W.: *A taxonomy of finite automata construction algorithms*, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993.
`citeseer.ist.psu.edu/watson94taxonomy.html`

Exercise

Example : let us have a set of the patterns P= $\{$tis, ti, iti$\}$:

- ☞ Create NFA that searches for P.
- ☞ Create DFA that corresponds to this NFA and minimize it. Draw the transition graphs of both the automata (DFA and the minimal DFA) and describe the procedure of minimization.
- ☞ Compare it to the result of the search engine SE.
- ☞ Solve the exercise using the algorithm of direct construction of DFA (by deriving) and discuss whether the result automata are isomorphic.

Derivation of RE by position vector I

Definition: <u>Position vector</u> is a set of numbers that correspond to the positions of those symbols of alphabet which can occur in the beginning of the tail of the string that is a part of the value of the given RE.

Example: let us have a regular expression:

$$a \quad . \quad b^* \quad . \quad c \tag{1}$$

To denote the position, we are going to use the wedge symbol $\wedge$. So the expression **(1)** is represented as:

$$\underset{\wedge}{a} \quad . \quad b^* \quad . \quad c \tag{2}$$

By deriving a denoted expression, we get a new denoted regular expression. The basic rule of derivation is this:

1. If the operand, by which we derive, is denoted, then we denote the positions right after this operand. Subsequently, we remove its denotation. It means that, by deriving the expression **(2)** by the operand a, we get:

$$a \quad . \quad \underset{\wedge}{b^*} \quad . \quad c \tag{3a}$$

Derivation of RE by position vector II

2. Since the construction, which generates also the empty string, is denoted, we denote the following construction as well:

$$a \quad . \quad \underset{\wedge}{b^*} \quad . \quad \underset{\wedge}{c} \tag{3b}$$

Now, by deriving by the operand b of the expression **(3b)**, we get:

$$a \quad . \quad b^* \quad . \quad \underset{\wedge}{c} \tag{4a}$$

3. Since the construction following the construction in iteration is denoted, the previous constructions have to be also denoted.

$$a \quad . \quad \underset{\wedge}{b^*} \quad . \quad \underset{\wedge}{c} \tag{4b}$$

By deriving the expression **(4b)** by the operand c, we get:

$$a \quad . \quad b^* \quad . \quad c \quad \underset{\wedge}{} \tag{5}$$

When a regular expression is denoted this way, it corresponds to the empty regular expression ε.

Derivation of RE by position vector III

- ☞ For every syntactic construction, we make a list of the starting positions at the initials of the members.
- ☞ If a construction symbol equals to the symbol we use for deriving, and it is located in the denoted position, then we move the denotation in front of the following position.
- ☞ If an iteration operator is located after the construction, and the denotation is at the end of the construction, then we append the list of the starting positions, which belong to this construction, to the resulting list.
- ☞ If the denotation is located before a construction, then we append the list of the starting positions of this construction to the resulting list.
- ☞ If the denotation is before the construction which generates also an empty string, then we append the list of the starting positions of the following construction to the resulting list.
- ☞ When we want to denote a construction inside parentheses, we must denote all the initials of the members inside the parentheses.

Derivation of RE by position vector: an example

Example: $a.b^*.c$, derived by a, b, c.

Part VII

Right-to-left search

Right-to-left search

Right-to-left search—principles.
Could the direction of the search be significant?
In which cases?

- ☞ *one pattern*—Boyer-Moore (BM, 1977), Boyer-Moore-Horspool (BMH, 1980), Boyer-Moore-Horspool-Sunday (BMHS, 1990)
- ☞ *n* patterns—Commentz-Walter (CW, 1979)
- ☞ an infinite set of patterns: *reversed regular expression*—Bucziłowski (BUC)

Boyer-Moore-Horspool algorithm

```
var: TEXT: array[1..T] of char;
    PATTERN: array[1..P] of char; I,J: integer; FOUND: boolean;
FOUND := false;    $I := P$;
while $(I \leq T)$ and not FOUND do
    $J := 0$;
    while $(J < P)$ and $(PATTERN[P - J] = TEXT[I - J])$ do
        $J := J + 1$;
    end while
    $FOUND := (J = P)$;

    if not FOUND then
        $I := I + SHIFT(TEXT[I - J], J)$
    end if
end while
```

$\texttt{SHIFT}(A, J) =$ **if** *A does not occur in the not yet compared part of the pattern* **then** $P - J$ **else** the smallest $0 \leq K < P$ such that $PATTERN[P - (J + K)] = A$;

When is it faster than KMP? When $O(T/P)$?
The time complexity $O(T+P)$.

Example: searching for the pattern `BANANA` in text
`I-WANT-TO-FLAVOR-NATURAL-BANANAS`.

CW algorithm

The idea: AC + right-to-left search (BM) [1979]

```
const LMIN=/the length of the shortest pattern/
var TEXT: array [1..T] of char; I, J: integer;
    FOUND: boolean;  STATE: TSTATE;
    g: array [1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F: set of TSTATE;
begin
 FOUND:=FALSE; STATE:=q0; I:=LMIN; J:=0;
 while (I<=T) & not (FOUND) do
  begin
   if g[STATE, TEXT[I-J]]=fail
    then begin I:=I+SHIFT[STATE, TEXT[I-J]];
               STATE:=q0; J:=0;
         end
    else begin STATE:=g[STATE, TEXT[I-J]]; J:=J+1 end
   FOUND:=STATE in F
  end
end
```

Construction of the CW search engine

INPUT: a set of patterns $P = \{v_1, v_2, \ldots, v_k\}$
OUTPUT: CW search engine
METHOD: *we construct the function g and introduce the evaluation of the individual states w*:

1. An initial state q_0; $w(q_0) = \varepsilon$.
2. Each state of the search engine corresponds to the suffix $b_m b_{m+1} \ldots b_n$ of a pattern v_i of the set P. Let us define $g(q, a) = q'$, where q' corresponds to the suffix $ab_m b_{m+1} \ldots b_n$ of a pattern v_i: $w(q) = b_n \ldots b_{m+1} b_m$; $w(q') = w(q)a$.
3. $g(q, a) = \utilde{\text{fail}}$ for every q and a, for which $g(q, a)$ was not defined in the step 2.
4. Each state, that correspond to the full pattern, is a final one.

CW—the function *shift*

Definition: $shift[STATE, TEXT[I-J]] = \min\left\{A, shift2(STATE)\right\}$, where $A = \max\left\{shift1(STATE), char(TEXT[I-J]) - J - 1\right\}$.

The functions are defined this way:

1. $char(a)$ is defined for all the symbols from the alphabet T as the least depth of a state, to that the CW search engine passes through a symbol a. If the symbol a is not in any pattern, then $char(a) = \text{LMIN} + 1$, where LMIN is the length of the shortest pattern. Formally: $char(a) = \min\left\{\text{LMIN}+1, \min\{d(q) | w(q) = xa,\ x \in T^*\}\right\}$.
2. Function $shift1(q_0) = 1$; for the other states, the value is $shift1(q) = \min\left\{\text{LMIN}, A\right\}$, where $A = \min\{k | k = d(q') - d(q)$, where $w(q)$ is its own suffix $w(q')$ and a state q' has higher depth than $q\}$.
3. Function $shift2(q_0) = \text{LMIN}$; for the other states, the value is $shift2(q) = \min\{A, B\}$, where $A = \min\{k | k = d(q') - d(q)$, where $w(q)$ is a proper suffix $w(q')$ and q' is a final state$\}$, $B = shift2(q') | q'$ is a predecessor of q.

CW—the function *shift*

Example: $P = \{cacbaa, aba, acb, acbab, ccbab\}$.

LMIN = 3,

	a	b	c	X
char	1	1	2	4

$w(q)$	*shift1*	*shift2*
ε	1	3
a	1	2
b	1	3
aa	3	2
ab	1	2
bc	2	3
ba	1	1
aab	3	2
aba	3	2
bca	2	2
bab	3	1
aabc	3	2
babc	3	1
aabca	3	2
babca	3	1
babcc	3	1
aabcac	3	2

Outline (week four)

1. Right-to-left search of an infinite set of patterns
2. Two-way jump automaton – a generalization of the so far learned left-to-right and right-to-left algorithms.
3. Hierarchy of the exact search engines.

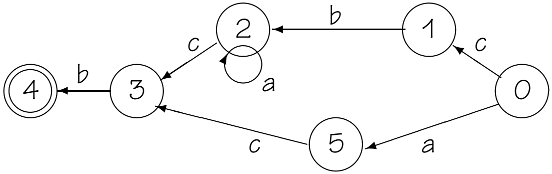
Part VIII

Search for an infinite set of patterns

Right-to-left search for an inf. set of patterns

Definition: ***reversed regular expression*** is created by reversion of all concatenation in the expression.

Example: reversed RE for $E = bc(a + a^*bc)$ is $E^R = (a + cba^*)cb$:


$$\begin{tikzcd}
 & 2 \arrow[loop below, "a"] \arrow[dl, "c"'] & & 1 \arrow[ll, "b"'] & \\
4 & 3 \arrow[l, "b"'] & & & 0 \arrow[ul, "c"'] \arrow[dl, "a"] \\
 & & 5 \arrow[ul, "c"] & &
\end{tikzcd}$$

Right-to-left search for an inf. set of patterns (cont.)

Bucziłowski: *we search for* E *such that we create* E^R and *we use it for determination of* $shift[STATE, SYMBOL]$ *for each state and undefined transition analogically as in the CW algorithm:*

	a	b	c	X
0		1		3 .
1	1		1	2 <u>(3!)</u> .
2		1		
3	1		1	1
4	1	1	1	1
5	1	1		1
	.	.		.

Two-way jump automaton I

Definition: ***2DFAS*** is $M = (Q, \Sigma, \delta, q_0, k, \uparrow, F)$, where

Q a set of states

Σ an input alphabet

δ a projection. $Q \times \Sigma \longrightarrow Q \times \{-1, 1, \ldots, k\}$

$q_0 \in Q$ an initial state

$k \in N$ max. length of a jump

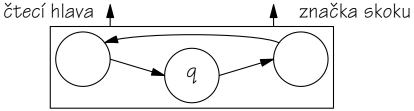
$\uparrow \notin Q \cup \Sigma$ a jump symbol

$F \subseteq Q$ a set of final states

Definition: ***a configuration of 2DFAS*** is a string of $\Sigma^* Q \Sigma^* \uparrow \Sigma^*$.

Definition: we denote ***a set of configurations 2DFAS M*** as $K(M)$.

Example: $a_1 a_2 \ldots a_{i-1}\ q\ a_i \ldots a_{j-1} \uparrow a_j \ldots a_n \in K(M)$:



Two-way jump automaton II

Definition: ***a transition of 2DFAS*** is a relation $\vdash \subseteq K(M) \times K(M)$ such that

- ☞ $a_1 \ldots a_{i-1} a_i\, q\, a_{i+1} \ldots a_{j-1} \uparrow a_j \ldots a_n \vdash a_1 \ldots a_{i-1}\, q'\, a_i a_{i+1} \ldots a_{j-1} \uparrow a_j \ldots a_n$ for $i > 1$, $\delta(q, a_{i+1}) = (q', -1)$ <u>(right-to-left comparison)</u>,
- ☞ $a_1 \ldots a_i\, q\, a_{i+1} \ldots a_{j-1} \uparrow a_j \ldots a_n \vdash a_1 \ldots a_i a_{i+1} \ldots a_{t-1}\, q' \uparrow a_t \ldots a_n$ for $\delta(q, a_{i+1}) = (q', m), \quad m \geq 1, \quad t = \min\{j + m, n + 1\}$ <u>(right-to-left jump)</u>,
- ☞ $a_1 \ldots a_j\, q\, a_{j+1} \ldots a_{i-1} \uparrow a_i \ldots a_n \vdash a_1 \ldots a_j a_{j+1} \ldots a_{t-1}\, q' \uparrow a_t \ldots a_n$ for $\delta(q, a_i) = (q', m), \quad m \geq 1, \quad t = \min\{i + m, n + 1\}$ <u>(left-to-right jump)</u>, .
- ☞ $a_1 \ldots a_{j-1}\, q\, a_j \ldots a_{i-1} \uparrow a_i a_{i+1} \ldots a_n \vdash a_1 \ldots a_{j-1}\, q'\, a_j \ldots a_{i-1} a_i \uparrow a_{i+1} \ldots a_n$ for $i > 1$, $\delta(q, a_i) = (q', 1)$ <u>(left-to-right comparison)</u>.

(Left-to-right rules are for the left-to-right engines and vice versa.)

Definition: $\vdash^k$, $\vdash^*$ analogically as in the SE.

Search engine hierarchy

Definition: ***the language accepted by the two-way automaton*** $M = (Q, \Sigma, \delta, q_0, k, \uparrow, F)$ is a set $L(M) = \{w \in \Sigma^* : q_0 \uparrow T \vdash^* w' f x w \uparrow,$ where $f \in F, w' \in \Sigma^*, x \in \Sigma\}$.

Theorem: $L(M)$ for *2DKAS* M is regular.
Example: formulate a right-to-left search of the pattern BANANA in the text I-WANT-TO-FLAVOUR-NATURAL-BANANAS using BM as 2DFAS and trace the search as a sequence of configurations of the 2DFAS.

Exercise

Let us have a regular expression $R = 1(0 + 1^*02)$ over the alphabet $A = \{0, 1, 2\}$.

- ☞ Construct a right-to-left DFA R (Bucziłowski) and compute the failure function. Draw the transition graph of this automaton including the failure function visualization.
- ☞ Express the resulting automaton as 2DFAS and trace searching in the text 11201012102.

Summary of the exact search

$$\begin{tikzcd}
 & \text{2DFAS} \arrow[dl] \arrow[dr] & & \\
\text{DFA} \arrow[d] & & \text{BUC} \arrow[d] & \infty \\
\text{AC} \arrow[d] & & \text{CW} \arrow[d] & k \\
\text{KMP} & & \text{BM} & 1 \\
\longrightarrow & & \longleftarrow & \text{direction — \# of patterns.}
\end{tikzcd}$$

Outline (Week five)

1. Fuzzy (proximity) search. Metrics for measurement of distance of strings.
2. Classification of search: 6D space of search problems.
3. Examples of creation of search engines.
4. Completion of the chapter about searching without text preprocessing.
5. Indexing basics.

Part IX

Proximity search

Metrics (for proximity search)

How to measure (metrics) the similarity of strings?

Definition: *we call* $d: S \times S \rightarrow R$ ***metrics*** if the following is true:

1. $d(x, y) \geq 0$
2. $d(x, x) = 0$
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, y) = 0 \Rightarrow x = y$ (identity of indiscernibles)
5. $d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality)

We call the values of the function d (distance).

Metrics for proximity search

Definition: let us have strings X and Y over the alphabet Σ. The minimal number of editing operation for transformation X to Y is

- ☞ ***Hamming distance***, R-distance, when we allow just the operation **R**eplace,
- ☞ ***Levenshtein distance***, DIR-distance, when we allow the operations **D**elete, **I**nsert and **R**eplace,
- ☞ ***Generalized Levenshtein distance***, $DIRT$-distance, when we allow the operations **D**elete, **I**nsert, **R**eplace and **T**ranspose. Transposition is possible at the neighbouring characters only.

They are ***metrics***, Hamming must be performed over strings of the same length, Levenshtein can be done over the different lengths.

Proximity search—examples

Example: Find such an example of strings X and Y, that simultaneously holds $R(X,Y)=5$, $DIR(X,Y)=5$, and $DIRT(X,Y)=5$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y, that holds simultaneously $R(X,Y)=5$, $DIR(X,Y)=4$, and $DIRT(X,Y)=3$, or prove the non-existence of such strings.

Example: find such an example of strings X and Y of the length $2n$, $n \in N$, that $R(X,Y)=2n$ and a) $DIR(X,Y)=2$; b) $DIRT(X,Y)=\lceil \frac{n}{2} \rceil$

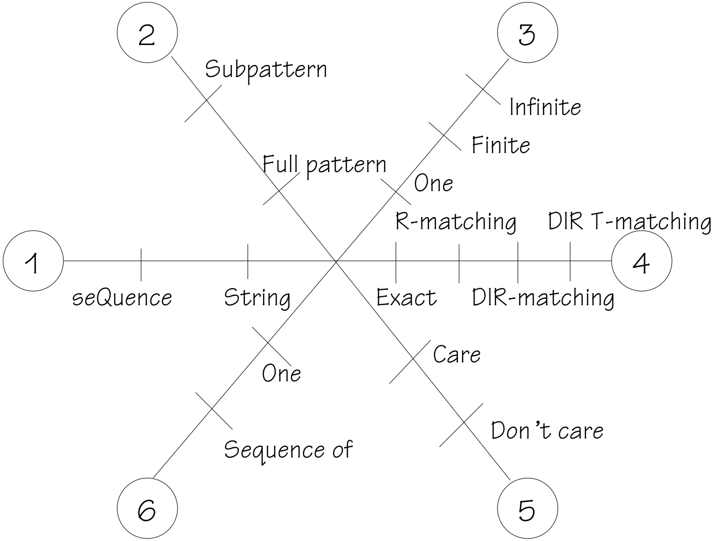
Classification of search problems

Definition: Let $T = t_1 t_2 \ldots t_n$ and pattern $P = p_1 p_2 \ldots p_m$. For example, we can ask:

1. is P a substring of T?
2. is P a subsequence of T?
3. is a substring or a subsequence P in T?
4. is P in T such that $D(P, X) \leq k$ for $k < m$, where $X = t_i \ldots t_j$ is a part of T (D is R, DIR or $DIRT$)?
5. is a string P containing ***don't care symbol*** $\emptyset$ (*****) in T?
6. is a sequence of patterns P in T?

Furthermore, the variants for more patterns, plus instances of the search problem yes/no, the first occurrence, all the overlapping occurrences, all the also non-overlapping occurrences.

6D classification of search problems [MEH] ([MAR])



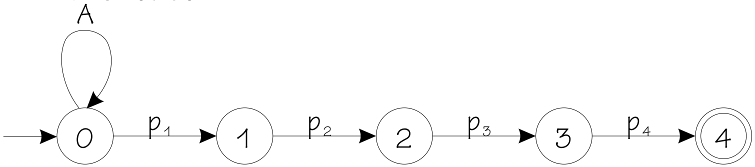
6D classification of search problems (cont.)

Dimension	1	2	3	4	5	6
	S	F	O	E	C	O
	Q	S	F	R	D	S
			I	D		
				G		

In total $2 \times 2 \times 3 \times 4 \times 2 \times 2 = 192$ search problems classified in a six-dimensional space.
For example, SFO??? denotes all the SE for search of one (entire) string.
For all these problems, we are going to learn how to create NFA for searching.

Examples of SE creation

Example: let $P = p_1 p_2 p_3 \dots p_m$, $m = 4$, A is any character of Σ.
NFA for SFOECO:


$$\begin{tikzcd}
\arrow[r] & 0 \arrow[loop above, "A"] \arrow[r, "p_1"] & 1 \arrow[r, "p_2"] & 2 \arrow[r, "p_3"] & 3 \arrow[r, "p_4"] & \boxed{4}
\end{tikzcd}$$

Search for a sequence of characters

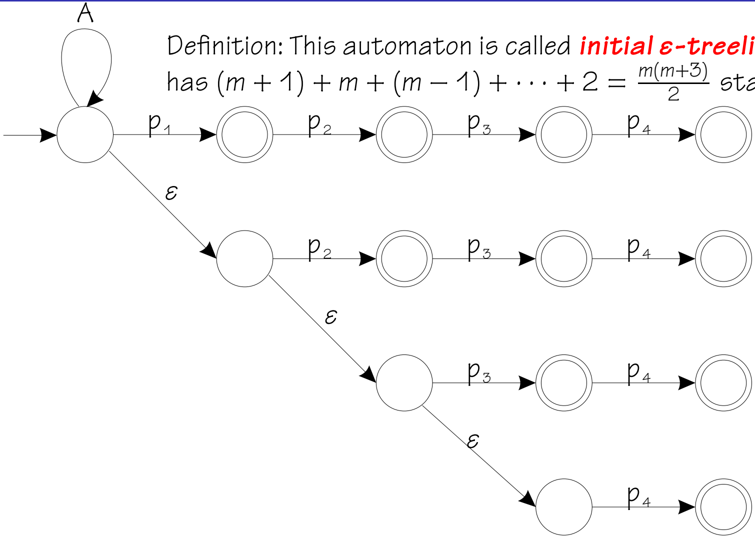
Example: NFA for QFOECO (se**Q**uence):

$$\begin{tikzcd}
{} \arrow[r] & 0 \arrow[r, "p_1"] \arrow[loop above, "A"] & 1 \arrow[r, "p_2"] \arrow[loop above, "\overline{p_2}"] & 2 \arrow[r, "p_3"] \arrow[loop above, "\overline{p_3}"] & 3 \arrow[r, "p_4"] \arrow[loop above, "\overline{p_4}"] & \boxed{4}
\end{tikzcd}$$

$\overline{p}$ is any character of Σ except for p. Automaton has $m + 1$ states for a pattern of the length m.

Search for a substring: NFA for SSOECO

Definition: This automaton is called ***initial ε-treelis*** and has $(m+1)+m+(m-1)+\cdots+2=\frac{m(m+3)}{2}$ states.



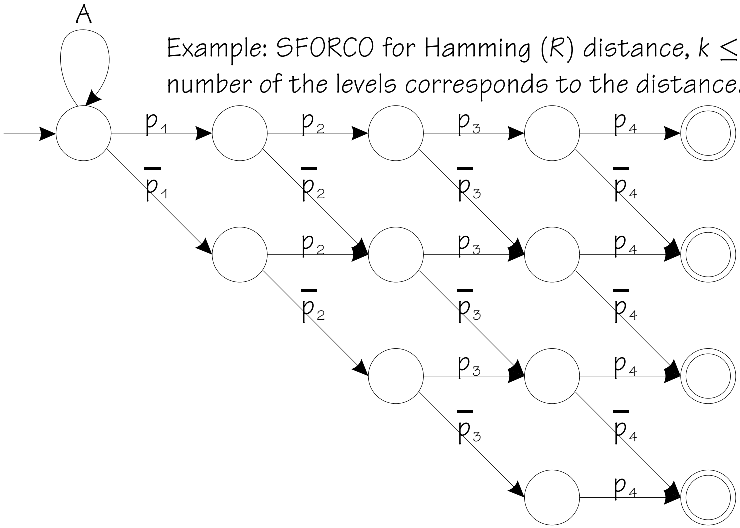
Search for a subsequence

Example: NFA for QSOECO is similar, *we just add some cycles for* non-matching characters *and* ε transitions to all *the existing* forward transitions (*or we concatenate the automaton* m-times).

Definition: Automaton for QSOECO is called ***ε-treelis***.

Proximity search of SFORCO

Example: SFORCO for Hamming (R) distance, $k \leq 3$: the number of the levels corresponds to the distance.



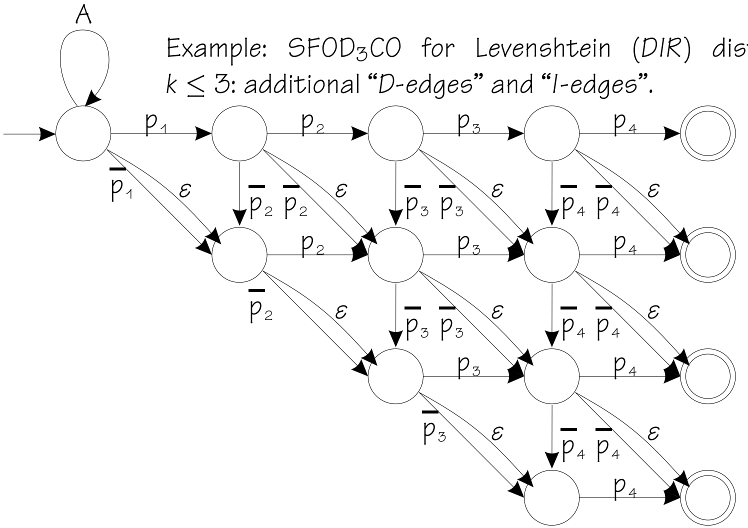
Proximity search of SFORCO

Definition: This automaton is called ***R-treelis***, and has $(m+1)+m+(m-1)+\cdots+(m-k+1)=(k+1)(m+1-\frac{k}{2})$ states.

The number of the level of the final state corresponds to the length of the found string from the pattern.

Proximity search of SFODCO for *DIR*-distance

Example: $SFOD_3CO$ for Levenshtein (*DIR*) distance, $k \leq 3$: additional "*D-edges*" and "*I-edges*".



SFOGCO

For the *DIRT*-distance, we add more new states to the SFODCO automaton that correspond to the operation of transposition and also the corresponding pair of edges for every transposition.

Animation program by Mr. Pojer for the discussed search automata is available for download from the course web page and is also installed in B311.

Simulation of NFA or determinisation? A hybrid approach.

The Prague Stringology Club and its conference series: see `http://www.stringology.org/`.

Outline (Week five)

1. Searching with text preprocessing; indexing methods.
2. Methods of indexing.
3. Automatic indexing, thesaurus construction.
4. Ways of index implementation.

Part X

Indexing Methods

Searching with text preprocessing

Large amount of texts? The text preprocessing!

- ☞ Index, indexing methods, indexing file, indexsequential file.
- ☞ Hierarchical structuring of text, ***tagging*** of text, ***hypertext***.
- ☞ Questions of word list storing (***lexicon***) and occurrence (hit) list storing, their updating.

Searching with text preprocessing

- ☞ granularity of the index items: *document – paragraph – sentence – word*

	word1	word2	word3	word4
doc1	1	1	0	1
doc2	0	1	1	1
doc3	1	0	1	1

- ☞ ***inverted file***, transposition

	doc1	doc2	doc3
word1	1	0	1
word2	1	1	0
word3	0	1	1
word4	1	1	1

Index searching

- ☞ Word order (primary key) in index $\longrightarrow$ ***binary search***
  Time complexity of one word searching in index: n index length, V pattern length
  $O(V \times \log_2(n))$
- ☞ searching for k words, pattern $p = v_1, \dots, v_k$
  $k \ll n \Rightarrow$ ***repeated binary search***
  s average pattern length, complexity?
  $O(s \times k \times \log_2 n)$
- ☞ As long as k and i are comparable: ***double dictionary method***.
- ☞ ***Hashing***.

However the speed $O(n)$ even $O(\log n)$ isn't usually sufficient, $O(1)$ is needed.

Implementation of indexing systems I

An appropriate choice of data structures and algorithms is for the index implementation crucial.

- ☞ Use of inverted file:

word1	1	0	1
word2	1	1	0
word3	0	1	1
word4	1	1	1

- ☞ Use of document list:

word1	1, 3
word2	1, 2
word3	2, 3
word4	1, 2, 3

- ☞ Coordinate system with pointers has 2 parts: a dictionary with pointers to the document list and a linked list of pointers to documents.

Indexing methods

- manual vs. automatic, pros/cons
- ***stop-list*** (words with grammatical meaning – conjunctions, prepositions, ...)
  1. not-driven
  2. driven (a special dictionary of words: indexing language assessment) – ***pass-list***, thesaurus.
- synonyms and related words.
- inflective languages: creating of registry with language support – ***lemmatization***.

Text analysis – choice of words for index

Frequency of word occurrences is for document identification significant.
English frequency dictionary:

1	the	69971	0.070	6	in	21341	0.128
2	of	36411	0.073	7	that	10595	0.074
3	and	28852	0.086	8	is	10099	0.088
4	to	26149	0.104	9	was	9816	0.088
5	a	23237	0.116	10	he	9543	0.095

☞ ***Zipf's law*** (principle of least resistance)
order $\times$ *frequency* $\cong$ *constant*

☞ ***Cumulative proportion of used words*** $CPW = \dfrac{\sum_{order=1}^{N} frequency_{order}}{text\ words\ count}$

☞ The rule 20–80: 20 % of the most frequent words make 80 % of text [MEL, fig. 4.19].

Automatic indexing method

Automatic indexing method is based on word significance derivation from word frequencies (cf. Collins-Cobuild dictionary); words with low and high frequency are cut out:
INPUT: n documents
OUTPUT: a list of words suitable for an index creation

1. We calculate a frequency $FREQ_{ik}$ for every document $i \in \langle 1, n \rangle$ and every word $k \in \langle 1, K \rangle$ [K is a count of different words in all documents].
2. We calculate $TOTFREQ_k = \sum_{i=1}^{n} FREQ_{ik}$.
3. We create a frequency dictionary for the words $k \in \langle 1, K \rangle$.
4. We set down a threshold for an exclusion of very frequent words.
5. We set down a threshold for an exclusion of words with a low frequency.
6. We insert the remaining words to the index.

Questions of threshold determination [MEL, fig. 4.20].

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 XI

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/…
…

$$\begin{array}{cccccc}\boxed{\begin{array}{c}\text{Computer}\\\text{(computational power)}\end{array}} & + & \boxed{\begin{array}{c}\text{human}\\\text{(inteligence)}\end{array}} & = & \boxed{\begin{array}{c}\text{valuable}\\\text{information}\end{array}} \\ & + & \text{time} & & \\ & + & \text{money} & & \end{array}$$

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?

- $\underbrace{\text{František Novák weighs 100 kg.}}$ $\longrightarrow$ **RDB**

  object property value — attribut1, attribut2, …
- František Novák likes beer. ? key value

  František Novák likes $\nearrow$

  Jana Novotná
- F. N. is an old honest man. $\longrightarrow$?

  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.

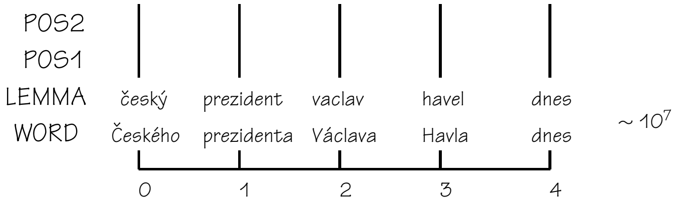
Corpora on FI

- 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, *n*th 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:

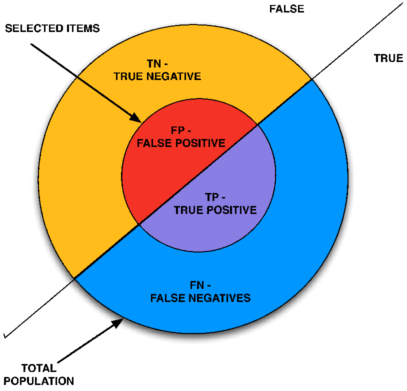
- The first is an independent inverted file, that contains a set of corpus positions.
- The second is an index of this file. This index returns for every code of item an input point belonging to an occurrence in inverted file.
- The third is a table of frequency of item code, which for each item code 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 $\boldsymbol{F_b = \frac{(b^2+1)PR}{b^2P+R}}$. ($F_0 = P$, $F_\infty = R$, 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 ten)

- ☞ 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 XII

Coding

Outline (Week eleven)

- ☞ 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 $S^+ \longrightarrow C^+$: $F(S_1 S_2 \dots S_k) = f(S_1)f(S_2)\dots f(S_k)$.
C^+ is sometimes called ***code***.

Basic properties of the code

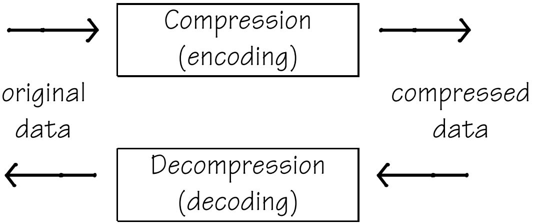
Definition: $x \in C^+$ is ***uniquely decodable*** regarding f, if there is maximum one sequence $y \in S^+$ so, that $f(y) = x$.
Definition: Code $K = (S, C, f)$ is ***uniquely decodable*** if all strings in C^+ are uniquely decodable.
Definition: A code is called a ***prefix*** one, if no code word 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) $E(Y) = \sum_{y \in Y} y p(y)$.

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 $\mathbf{S}$ is p_i for $i = 1, \ldots, n$, $n \in N$.

Definition: ***Entropy of information content of unit $\mathbf{x_i}$*** (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 $X = x_{i_1} x_{i_2} \dots x_{i_k} \in \mathbf{S}^+$ 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: $R(X) = l(X) - H(X) = \sum_{j=1}^{k}(d_{i_j} + \log_2 p_{i_j})$ 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 S*** is

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

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

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

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 $F(32) = 00101011$.
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* $\alpha(N) = \underbrace{00\ldots0}_{N-1}1.$
- 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 $\alpha(|\beta(N)|)$.
- *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

- $\gamma'(N) = \alpha(|\beta(N)|)\beta'(N)$ 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.
```

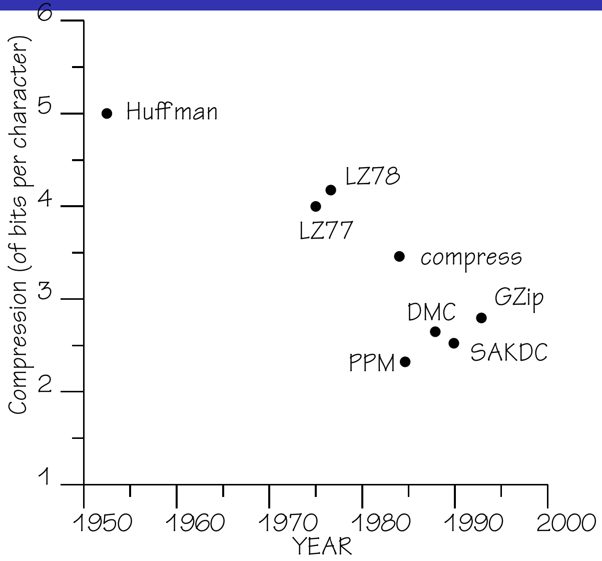
Data compression – introduction

- ☞ 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* $\underline{e}$ 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), $P(a|x_1 x_2 \dots x_n)$
- 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 twelwe)

- ☞ 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 $|S| \geq 2$ 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
```

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* ⇔ 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.
- ☞ $\mathrm{AL}_{HD} \leq 2\mathrm{AL}_{HS}$.
- ☞ Vitter $\mathrm{AL}_{HD} \leq \mathrm{AL}_{HS} + 1$.
- ☞ 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

Dictionary	Compressed data

Compressed dictionary	Compressed data

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 $\ldots$. If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

Outline (Week thirteen)

- 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

a	b	c	b	a	b	b	a	a	b	a	c	b

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

not enc. part ($|B| \sim$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.

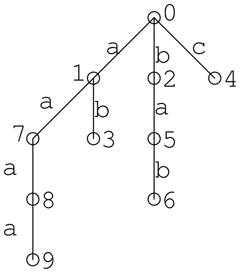
LZ78 – example

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 $y(t) = t - \text{LAST}(x_t)$;
        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 fourteen)

- ☞ 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 $\longrightarrow$ 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*** $\boldsymbol{W = (w_{ij})}$ of type m, n,

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

Query $\boldsymbol{Q = (q_1, \ldots, q_n)}$

- $S(Q, D_i) = \sum_i q_i w_{ij}$ ***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`