

PVO30 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2011

Part I

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**, $f : S \rightarrow C^+$ is an injective mapping. f can be expanded to $S^+ \rightarrow C^+$: $f(S_1 S_2 \dots S_k) = f(S_1) f(S_2) \dots f(S_k)$. C^+ is sometimes called **code**.

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**, $f: S \rightarrow C^+$ is an injective mapping. f can be expanded to $S^+ \rightarrow C^+$: $f(S_1S_2 \dots S_k) = f(S_1)f(S_2) \dots f(S_k)$. C^+ is sometimes called **code**.

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**, $f: S \rightarrow C^+$ is an injective mapping. f can be expanded to $S^+ \rightarrow C^+$: $F(S_1S_2 \dots S_k) = f(S_1)f(S_2) \dots f(S_k)$. 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 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 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 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 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 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 $|C| = 2$.

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 $|C| = 2$.

Basic properties of code

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

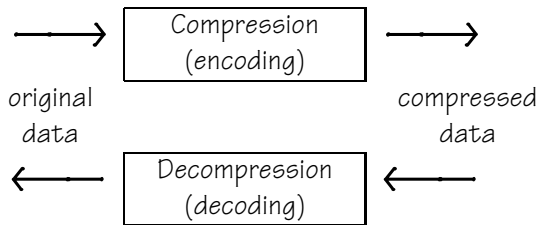
Example: block ? prefix

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*):

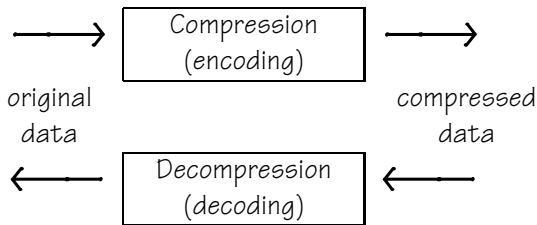


Definition: **Compression ratio** is a ratio of length of compressed data and length of original data.

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

Compression and decompression

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

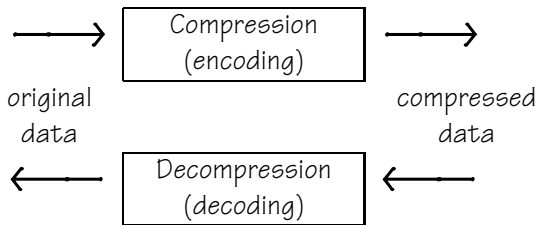


Definition: **Compression ratio** is a ratio of length of compressed data and length of original data.

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

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 and 4, and rarely 5 and 6.

Entropy and redundancy I

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

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

Definition: **Entropy of information content of unit x_i** (measure of amount of information or uncertainty) is $H(x_i) = H_i = -\log_2 p_i$ bits. A source unit with more probability bears less information.

Entropy and redundancy II

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

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

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

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

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

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

Entropy and redundancy II

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

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

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

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

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

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

Entropy and redundancy II

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

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

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

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

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

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

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see <ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kod>

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

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

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

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

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see <ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kod>

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

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

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

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

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see <ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kod>

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

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

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

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

Entropy a redundancy III

Axiomatic introduction of entropy see [MAR], details of derivation see <ftp://www.math.muni.cz/pub/math/people/Paseka/lectures/kod>

Definition: $R(X) = l(X) - H(X) = \sum_{j=1}^k (d_{ij} + \log_2 p_{ij})$ 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

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

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

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

Entropy and redundancy 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 $c_1, c_2 \in \mathbb{R}$ 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**

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

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

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

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

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 $a(N) = \underbrace{00 \dots 0}_{N-1} 1$.

☞ 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 $a(|\beta(N)|)$.

☞ example: $\gamma(6) = 0\bar{1}0\bar{0}\bar{1}$

☞ $C_\gamma = \{\gamma(N) : N > 0\} = (O\{0, 1\})^*1$ is regular and therefore it's decodable by finite automaton.

The universal coding of the integers III

- ☞ $\gamma'(N) = a(|\beta(N)|)\beta'(N)$ the same length (bit permutation $\gamma(N)$), but more readable
- ☞ $C_{\gamma'} = \{\gamma'(N) : N > 0\} = \{0^k 1 \{0, 1\}^k : k \geq 0\}$ is not regular and the decoder needs a counter
- ☞ $\delta(N) = \gamma(|\beta(N)|)\beta'(N)$
- ☞ example: $\delta(4) = \gamma(3)00 = 01100$
- ☞ decoder δ : $\delta(?) = 0011?$
- ☞ ω :
 $K := 0$;
while $\lfloor \log_2(N) \rfloor > 0$ **do**
 begin $K := \beta(N)K$;
 $N := \lfloor \log_2(N) \rfloor$
 end.

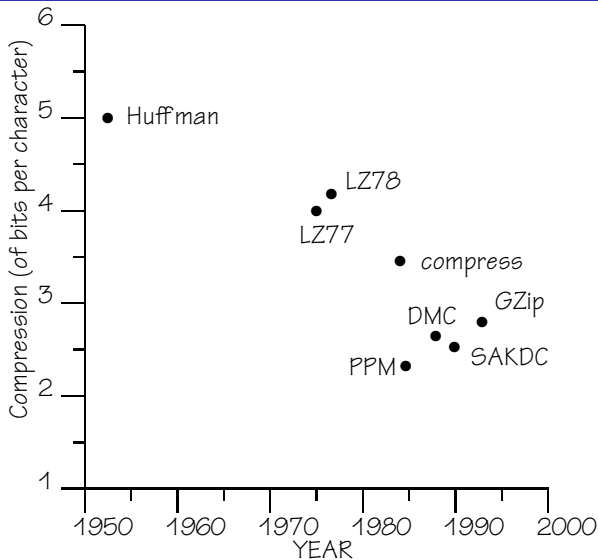
Data compression – introduction

- ☞ Information encoding for communication purposes; Despite tumultuous evolution of capacities for data storage, there is still a lack of space. Redundancy → 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 encoding.

Data compression – evolution

- ☞ 1838 Morse, code e by frequency.
- ☞ 1949 Shannon, Fano, Weaver.
- ☞ 1952 Huffman; 5 bits per character.
- ☞ 1979 Ziv-Lempel; **compress** (Roden, Welsh, Bell, Knuth, Miller, Wegman, Fiala, Green, ...); 4 bits per character.
- ☞ eighties and nineties PPM, DMC, **gzip** (zlib), SAKDC; 2–3 bits/character
- ☞ at the turn of millennium **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 throughpasses, necessity of model portage)
- ☞ adaptive modeling (1. throughpass, 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), $P(a|x_1x_2 \dots x_n)$
- ☞ 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 eleven)

- ☞ Huffman coding.
- ☞ Adaptive Huffman coding.
- ☞ Arithmetic 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_cXC_c generalization to any repetitious character $\$*****55 \rightarrow \S_c*655
- ☞ MNP Class 5 RLE – $CXXX DDDDDBBAAAA \rightarrow 5DDDBB4AAA$
- ☞ 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 isn't optimal, two passes of encoder through text, static \rightarrow

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  $S_1$  and  $S_2$  so, that both
```

```
        sequences have roughly the same total probability;
```

```
        add to all code words from  $S_1$  0;
```

```
        add to all code words from  $S_2$  1;
```

```
        SF-SPLIT( $S_1$ ); SF-SPLIT( $S_2$ );
```

```
    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 \rightarrow preparation in advance.

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

Huffman coding – sibling property

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

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

Huffman coding – properties of Huffman trees

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

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

Adaptive Huffman coding

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

Principle of arithmetic coding

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

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.

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 through text)
- adaptive

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 through text)
- adaptive

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 through 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 consists of about 150 most frequent words)

Disadvantages:

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

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 consists of about 150 most frequent words)

Disadvantages:

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

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 consists 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 data (the dictionary is a small part of data – corpora; CQP).

Semiadaptive dictionary methods

Dictionary	Compressed data
------------	-----------------

Compressed dictionary	Compressed data
-----------------------	-----------------

Advantages: extensive data (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 $N = 1, 2, \dots$
- 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 its components of $(K - 1)$ -grams, $(K - 2)$ -grams \dots . If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

Semiadaptive dictionary methods – dictionary creation procedure

- 1 The frequency of N -grams is determined for $N = 1, 2, \dots$
- 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 its components of $(K - 1)$ -grams, $(K - 2)$ -grams If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

Semadaptive dictionary methods – dictionary creation procedure

- 1 The frequency of N -grams is determined for $N = 1, 2, \dots$
- 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 its components of $(K - 1)$ -grams, $(K - 2)$ -grams \dots . If, by reducing of frequencies, a frequency of a component is greatly reduced, then it's excluded from the dictionary.

Outline (Week eleven)

- 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 – sliding window methods

LZ78 – methods of increasing dictionary



encoded part

(window, $N \leq 8192$)

not enc. part

($|B| \sim 10-20b$)

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 (l, J, A) , where l 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.

Adaptive dictionary methods

LZ77 – sliding 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

($|B| \sim 10-20b$)

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 (l, J, A) , where l 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.

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.

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.

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.

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)

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)

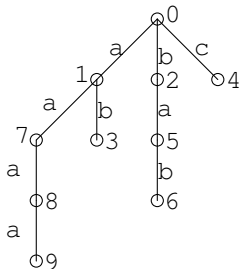
Methods with increasing dictionary

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.

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 → 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, \dots, a_n , input sequence contains x_1, x_2, \dots, x_m . The value $\text{LAST}(a_i)$ containing the interval from last occurrence is initialized to zero.

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

Sequence y_1, y_2, \dots, 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.

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 eleven)

- ☞ 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.
- ☞ Compression using neural networks.

Querying and TIS models

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

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

Blair's query tuning

The search lies in reducing of uncertainty

- 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.

Blair's query tuning

The search lies in reducing of uncertainty

- 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*.

Blair's query tuning

The search lies in reducing of uncertainty

- 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 is half 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

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, \dots, a_n be terms, D_1, \dots, D_m documents, and **relevance matrix** $W = (w_{ij})$ of type m, n ,

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

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

- $S(Q, D_i) = \sum_j q_j w_{ij}$ **similarity coefficient**
- $\text{head}(\text{sort}(S(Q, D_i)))$ 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: $\frac{TD}{\sqrt{\sum_j TD_j^2}}$
- normalization of weights for query: $\left(\frac{1}{2} \times \frac{\frac{1}{2}TF}{\max TF_i}\right) \times \log_2 \frac{m}{k}$

[POK, pages 85–113].

Automatic structuring of texts

- ☞ Interrelations *between documents* in TIS.
- ☞ Encyclopedia (OSN, Funk and Wagnalls New Encyclopedia).
- ☞ [SBA]
`http://columbus.cs.nott.ac.uk/compsci/epo/epodd/ep056gs`
- ☞ Google/CiteSeer: „automatic structuring of text files“

Similarity of documents

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

Lexicon storage



[MeM] Mehryar Mohri: On Some Applications of Finite-State Automata Theory to Natural Language Processing, *Natural Language Engineering*, 2(1):61–80, 1996.

<http://www.research.att.com/~mohri/cl1.ps.gz>

Signature methods

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

- ☞ *chained*

- ☞ *layerd*

More [POK, pages 65–76], see [MAR].

List of materials in U Marečka store

[MAR] Materials for subject PVO30 to copy in the bookstore U Marečka and in materials of the subject in IS.

- 1 Slides of lectures of the subject, 4 slides per A4 sheet.
- 2 copy [MEL].
- 3 copy [POK].
- 4 article [MEH].
- 5 materials about Google [GOO] (plus Czech summary).
- 6 chapter 5.7 from [KOR] (similarity).
- 7 [MeM].
- 8 other (NLP, diagram with compress algorithms,...).