HY463 - Συστήματα Ανάκτησης Πληροφοριών
Information Retrieval (IR) Systems

Συμπίεση Κειμένου
Text Compression

Γιάννης Τζίτζικας

Διάλεξη : 14b
Ημερομηνία :

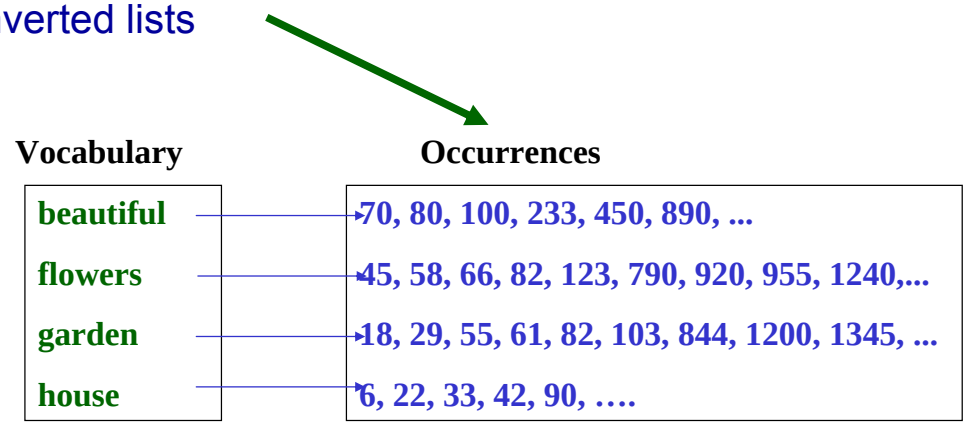
Διάρθρωση Διάλεξης

- Συμπίεση Ανεστραμμένου Ευρετηρίου
- Εισαγωγή στη συμπίεση
- Βασικές Έννοιες
- **Στατιστικές** Τεχνικές Συμπίεσης
- Τεχνικές Συμπίεσης **Λεξικού** (Dictionary)

- *Σχετικό μάθημα: HY438 (Συμπίεση Δεδομένων και Σημάτων)*

Inverted File Compression (Sec. 7.4.5 of MIR book)

- An Inverted file contains:
  - (a) a vocabulary containing all distinct words in the text collection
  - (b) for each word in the vocabulary, a list of all documents in which that word occurs
- The size of the inverted file can be reduced by compressing the inverted lists



Inverted File Compression: Compressing Inverted Lists

Occurrences

70, 80, 100, 233, 450, 890, ...

45, 58, 66, 82, 123, 790, 920, 955, 1240,...

18, 29, 55, 61, 82, 103, 844, 1200, 1345, ...

6, 22, 33, 42, 90, ….

- As the list of document numbers within the inverted list is in ascending order, it can also be considered as a **sequence of *gaps* between document numbers**.
  - E.g. [2,8,22,30] -> [2,6,14,8]
  - [21002, 21008, 21022, 21030] -> [21002,6,14,8]
- Since processing is usually done sequentially starting from the beginning of the list, the original document numbers can always be recomputed through sums of the gaps.

Inverted File Compression: Compressing Inverted Lists (II)

- These gaps are
  - **small** for frequent words and
  - **large** for infrequent words
- Compression can be obtained by encoding small values with shorter codes

- Codings
  - Unary code
    - An integer x is coded as (x-1) one bits followed by a zero bit, so the code for the integer 3 is 110

Inverted File Compression: Compressing Inverted Lists (III)

- Elias-γ
  - the number x is represented by a concatenation of two parts:
    - (1) a unary code for $1 + \lfloor \log x \rfloor$ and
    - (2) a code of $\lfloor \log x \rfloor$ bits that represents the values of $x - 2^{\lfloor \log x \rfloor}$ in binary

$1 = 2^0 + 0 = 1$
$2 = 2^1 + 0 = 010$
$3 = 2^1 + 1 = 011$
$4 = 2^2 + 0 = 00100$
$5 = 2^2 + 1 = 00101$
$6 = 2^2 + 2 = 00110$
$7 = 2^2 + 3 = 00111$
$8 = 2^3 + 0 = 0001000$
$9 = 2^3 + 1 = 0001001$
$10 = 2^3 + 2 = 0001010$
$11 = 2^3 + 3 = 0001011$
$12 = 2^3 + 4 = 0001100$
$13 = 2^3 + 5 = 0001101$
$14 = 2^3 + 6 = 0001110$
$15 = 2^3 + 7 = 0001111$
$16 = 2^4 + 0 = 000010000$
$17 = 2^4 + 1 = 000010001$

Inverted File Compression: Compressing Inverted Lists (IV)

- Elias-δ
  - represents the prefix indicating the number of binary bits by the Elias-γ code

$1 = 2^0 \Rightarrow N' = 0, N = 1 \Rightarrow 1$
$2 = 2^1 + 0 \Rightarrow N' = 1, N = 2 \Rightarrow 0100$
$3 = 2^1 + 1 \Rightarrow N' = 1, N = 2 \Rightarrow 0101$
$4 = 2^2 + 0 \Rightarrow N' = 2, N = 3 \Rightarrow 01100$
$5 = 2^2 + 1 \Rightarrow N' = 2, N = 3 \Rightarrow 01101$
$6 = 2^2 + 2 \Rightarrow N' = 2, N = 3 \Rightarrow 01110$
$7 = 2^2 + 3 \Rightarrow N' = 2, N = 3 \Rightarrow 01111$
$8 = 2^3 + 0 \Rightarrow N' = 3, N = 4 \Rightarrow 00100000$
$9 = 2^3 + 1 \Rightarrow N' = 3, N = 4 \Rightarrow 00100001$
$10 = 2^3 + 2 \Rightarrow N' = 3, N = 4 \Rightarrow 00100010$
$11 = 2^3 + 3 \Rightarrow N' = 3, N = 4 \Rightarrow 00100011$
$12 = 2^3 + 4 \Rightarrow N' = 3, N = 4 \Rightarrow 00100100$
$13 = 2^3 + 5 \Rightarrow N' = 3, N = 4 \Rightarrow 00100101$
$14 = 2^3 + 6 \Rightarrow N' = 3, N = 4 \Rightarrow 00100110$
$15 = 2^3 + 7 \Rightarrow N' = 3, N = 4 \Rightarrow 00100111$
$16 = 2^4 + 0 \Rightarrow N' = 4, N = 5 \Rightarrow 00101000$
$17 = 2^4 + 1 \Rightarrow N' = 4, N = 5 \Rightarrow 00101001$

- Golomb
  - presented another run-length coding method for positive integers. It is very effective when the probability distribution is geometric.
- Example codes for integers:
  - MIR BOOK page 185

Searching Compressed Files **Inverted Files**

- Επανάληψη
  - we represent gaps by schemes that favor small numbers
    - If we first cluster the documents and reassign to them document identifiers, then we have more small gaps => more space savings!
  - reductions in 90% can be obtained by block addressing indices with blocks of 1 Kb size
- Remarks:
  - Compression does not necessarily degrade time performance
    - most of the time spent in answering a query is in the disk transfer
  - Query times on compressed or decompressed indices are roughly similar

Clustering and Compression of Inverted Files

If we first cluster the documents and reassign to them document identifiers, then we have more small gaps => more space savings!

Διαβάστε το άρθρο: ECIR'2007 Best Paper Award
(υπάρχει στο wiki)

Searching Compressed Files

- Huffman coding allows searching directly on compressed text
  - *(we will describe Huffman coding later on)*
- Since Huffman coding needs to store the codes of each symbol, this scheme has to store the whole vocabulary of the corpus
  - If we consider words as symbols, then they are already stored in the vocabulary of the inverted index)
- Evaluating single word queries:
  - they are first searched in the vocabulary
  - their (Huffman) codes are collected which are then searched in the compressed file

COMPRESSION

Εισαγωγή

- *Encoding* transforms data from one representation to another
- *Compression* is an encoding that takes less space
- *Lossless*: decoder can reproduce message exactly
- *Lossy*: can reproduce message approximately
- *Degree of compression*: (Original - Encoded) / Encoded
  - example: (125 MB - 25 MB) / 25 MB = 400%
- *Compression ratio*: the size of the compressed file as a fraction of the uncompressed file
  - example: 25MB/125 MB = 0.2
    - » (compressed size) = 0.2 (original size)

Συμπίεση

- Advantages of Compression
  - Save space in memory (e.g., compressed cache)
  - Save space when storing (e.g., disk, CD-ROM)
  - Save time when accessing (e.g., I/O)
  - Save time when communicating (e.g., over network)
- Disadvantages of Compression
  - Costs time and computation to compress and uncompress
  - Complicates or prevents random access
  - May involve loss of information (e.g., JPEG)
  - Makes data corruption *much* more costly. Small errors may make all of the data inaccessible.

Παραδείγματα Τεχνικών Συμπίεσης

Generic File Compression
- files: gzip, bzip, BOA
- archivers: ARC, PKZip
- file systems: NTFS

Communication
- Fax: ITU-T Group 3
- Modems: V.42bis protocol, MNP5

Multimedia
- Images: gif, jbig, jpeg-ls, jpeg
- TV: HDTV (mpeg-4)
- Sound: mp3

Συμπίεση Κειμένου

- Text Compression vs Data Compression
  - Text compression predates most work on general data compression.
  - Text compression is a kind of data compression optimized for text (i.e., based on a language and a language model).
  - Text compression can be faster or simpler than general data compression, because of assumptions made about the data.
  - Text compression assumes a language and language model;
  - Data compression learns the model on the fly.
  - Text compression is effective when the assumptions are met;
  - Data compression is effective on almost any data with a skewed distribution («ασύμμετρη κατανομή»)

Διάκριση Τεχνικών Συμπίεσης

(A) **Στατιστικές** τεχνικές (statistical)

- βασίζονται σε εκτιμήσεις της πιθανότητας εμφάνισης των συμβόλων
- όσο πιο ακριβείς είναι αυτές οι εκτιμήσεις τόσο καλύτερη συμπίεση επιτυγχάνεται
- παραδείγματα τέτοιων τεχνικών:
  - Huffman coding
  - Arithmetic coding

(B) Τεχνικές **βάσει Λεξικού** (dictionary-based)

- αντικαθιστούν μια ακολουθία συμβόλων με έναν δείκτη προς μια προηγούμενη εμφάνιση της ακολουθίας
- παραδείγματα τέτοιων τεχνικών:
  - Ziv-Lempel family
    - They can compress English text to less than 4 bits per character

Βασικές Έννοιες

- A ***symbol*** can be a character, a text word, or a fixed number of characters.
- ***Alphabet***: the set of all possible symbols in the text
- ***Modeling***: the task of estimating the probability of each next symbol
- ***Model:*** a collection of probability distributions, one for each context in which a symbol can be coded
- ***Coding*****:** The conversion of symbols to binary digits
- ***Decoding*****:** Reconstruction of the original text (using the same model)

(A) Στατιστικές Τεχνικές: Εισαγωγή (I)

Huffman Coding

- Ιδέα:
  - Κωδικοποιεί με λιγότερα bits τα σύμβολα με μεγάλη πιθανότητα εμφάνισης
- Αποτελεσματικότητα:
  - They are able to compress English text to approximately 5 bits per character (instead of the usual 7-8)
- Word-based Huffman
  - They are able to compress English text to approximately 2 bits per character

Arithmetic Coding

- Ιδέα:
  - Computes the code incrementally, one symbol at a time, as opposed to Huffman coding scheme in which each different symbol is pre-encoded using a fixed-length number of bits.
- Αποτελεσματικότητα
  - They can compress English text to just over 2 bits per character
- Αδυναμία
  - The incremenal nature does not allow decoding a string which starts in the middle of the compressed file. This requires decoding the whole text from the beginning until the desired word. This makes arithmetic coding inadequate for use in IR environment.

The Lower Bound of Compression

In an optimal encoding scheme,
a symbol that is expected to occur with probability p
should be assigned a code of length $\log_2 1/p$ bits.
[Shannon]

- The number of bits in which a symbol is best coded represents the *information content* of the symbol

- Παραδείγματα
  - p=1 → $\log_2 1/1 = 0$
  - p=1/2 → $\log_2 1/(1/2) = \log_2 2 = 1$
  - p=1/4 → $\log_2 1/(1/4) = \log_2 4 = 2$
  - Έστω A(1/2), B(1/4), C(1/4)
    - |code(A)|=1, |code(B)|=2, |code(C)|=2
    - For example: code(A)="1", code(B)="00", code(C)="01"

Στατιστικές Τεχνικές: The Lower Bound of Compression (II)

- The average amount of information per symbol over the whole alphabet is called the ***entropy*** of the probability distribution, given by:
  - $E = \Sigma\ pi\ \log_2 1/pi$
- E is a *lower bound* on compression, measured in bits per symbol, which applies to any coding method based on the probability distribution pi.

Παράδειγμα

- A(1/2), B(1/4), C(1/4), cod(A)=1, code(B)=00, code(C)=01

  E = 1/2*1 + 1/4*2 + 1/4*2 = **1.5**
- A(1/2), B(1/2) cod(A)=1, code(B)=0

  E = 1/2*1 + 1/2*1 = **1**

Στατιστικές Τεχνικές: Modeling

- Σκοπός
  - provide a probability assignment for the next symbol to be coded.
  - High compression can be obtained by forming good models
- Διάκριση μοντέλων
  - (m1) Adaptive
  - (m2) Static
  - (m3) Semi-static

Στατιστικές Τεχνικές>Modeling: (m1) **Adaptive Models**

- Start with no information about the text and progressively learn about its statistical distribution as the compression process goes on
- Thus, adaptive models need only one pass over the text and store no additional information apart from the compressed text
- For long enough texts, these models converge to the true statistical distribution of the text
- Disadvantage:
  - The decompression of a file has to start from its beginning (since information on the distribution of the data is stored incrementally inside the file)
  - Inadequate for full-text retrieval where random access to compressed patterns is a must

Στατιστικές Τεχνικές>Modeling: (m2) **Static Models**

- They assume an average distribution for all input texts
- The modeling phase is done only once for all texts to be coded in the future
- They tend to achieve poor compression ratios when the data deviates from initial statistical assumptions
  - e.g. a model adequate for English literary texts will probably perform poorly for financial texts containing a lot of different numbers, as each number is relatively rare and so receives long codes

Στατιστικές Τεχνικές>Modeling: (m3) **Semi-static Models**

- They do not assume any distribution on the data, but learn it in a first pass.
- In a second pass, they compress the data using a fixed code derived from the distribution learned from the first pass.
- At decoding time, information on the data distribution is sent to the decoder before transmitting the encoded symbols.
- Disadvantages:
  - they must make 2 passes
  - the information on the data distribution must be stored to be used by the decoder to decompress
- Advantage for IR:
  - since the same codes are used at every point in the compressed file, direct access is possible.

Στατιστικές Τεχνικές>Modeling: **Word-based Models**

- They take words instead of characters as symbols.
- Advantages of IR:
  - they achieve higher compression rates
  - words are the atoms on which most IRS are built
  - words are already stored for indexing purposes (inverted files) and so might be used as part of the model for compression
  - word frequencies are also useful in answering queries involving combinations of words because the best strategy is to start with the least frequent words first

Στατιστικές Τεχνικές> **Coding**

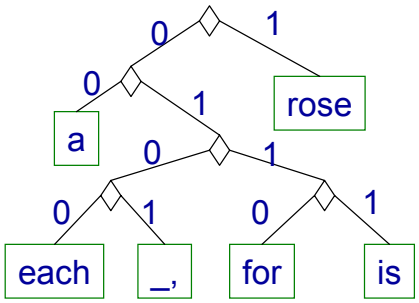
- Coding is the task of obtaining the representation (code) of a symbol based on a probability distribution given by a model.
- Design goals
  - assign short codes to likely codes and long codes to unlikely ones
  - coding and decoding speed
- As the entropy of a probability distribution is a lower bound on how short the average length of a code can be, the quality of a coder is measured in terms of how close to the entropy it is able to get

Στατιστικές Τεχνικές> **Coding**
Semi-Static Huffman

- First pass: the modeler determines the probability distribution of the symbols and builds a coding tree
- Second pass: each next symbol is encoded according to the coding tree
- Compression is achieved by assigning shorter codes to more frequent symbols.
  - Huffman codes
    - Invented by Huffman as a class assignment in 1950.
    - Used in many (if not most) compression algorithms: gzip, bzip, jpeg (as option), fax compression,…
- Decompression uniqueness is guaranteed because no code is a prefix of another

Στατιστικές Τεχνικές> Coding
Semi-Static Huffman: Example

- Text: «**for each rose, a rose is a rose**»
- Frequencies: «rose»(3), «a»(2), «for»(1), «each»(1), « **,**»(1), «**is**»(1)
- Huffman tree: binary trie built on binary codes



No code is prefixof another

- Original text: for each rose, a rose is a rose
- Compressed text: 0110 0100 1 0101 00 1 0111 00 1

Στατιστικές Τεχνικές> Coding
Semi-Static Huffman: Building the Huffman Tree

(1) For each symbol of the alphabet a node containing the symbol and its probability is created

At this point we have a forest of one-node trees whose probabilities sum up to 1

(2) The two nodes with the smallest probabilities become children of a newly created parent node. To this node with associate the sum of the probabilities of its children

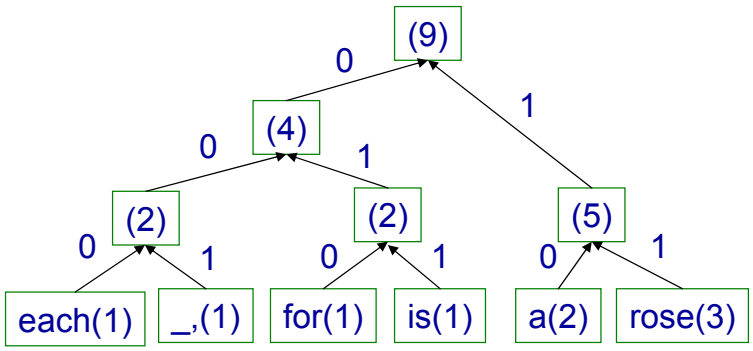
(3) The operation is repeated ignoring nodes that are already children, until there is only one node which becomes the root of the tree.

Notes:

- By delaying the pairing of nodes with high probabilities, the algorithm necessarily places them closer to the root node, making their code smaller
- The two branches from every internal node are consistently labeled 0 and 1
- Given s symbols and their frequencies in the text, the algorithm build the Huffman tree in O(s log s) time.

Στατιστικές Τεχνικές> Coding
Semi-Static Huffman: Building the Huffman Tree

- Text: «**for each rose, a rose is a rose**»
- Frequencies: «rose»(3), «a»(2), «for»(1), «each»(1), « **,**»(1), «**is**»(1)



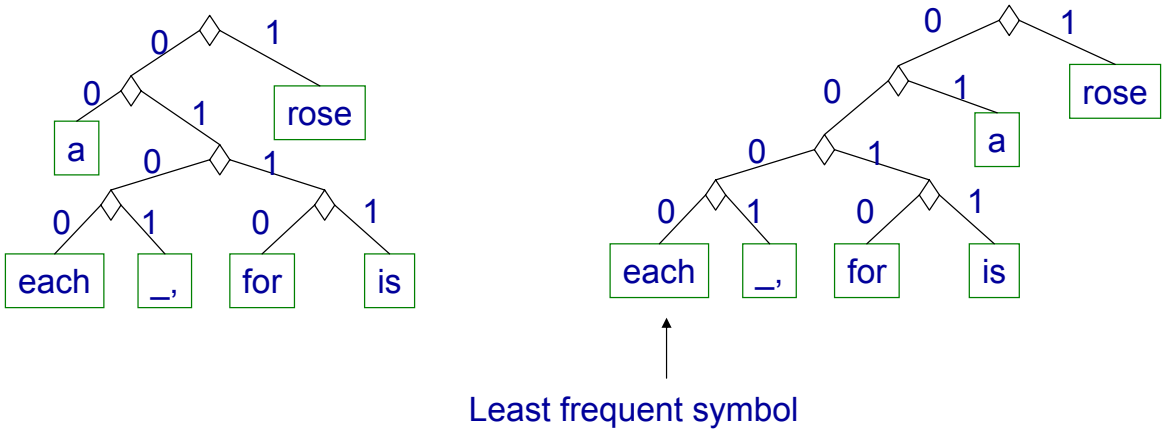
each:	000
_,:	001
for:	010
is:	011
a:	10
rose:	11

Στατιστικές Τεχνικές> Coding
Semi-Static Huffman: Canonical Tree

- Motivation:
  - The number of Huffman trees which can be built for a given probability distribution is large:
  - This is because interchanging left and right subtrees of any internal node results in a different tree whenever the two subtrees are different in structure, but the weighted average code length is not affected
  - Instead of using any kind of tree, the preferred choice for most applications is to adopt a **canonical tree** which imposes a particular order to the coding bits.

Semi-Static Huffman: Canonical Tree (II)

- A Huffman tree is canonical when the height of the left subtree of any node is never smaller than that of the right subtree, and all leaves are in increasing order of probabilities from left to right



Semi-Static Huffman: Encoding & Decoding

Encoding: Start at leaf of Huffman tree and follow path to the root. Reverse order of bits and send.

Decoding: Start at root of Huffman tree and take branch for each bit received. When at leaf can output message and return to root

The stream of bits in the compressed file is traversed from left to right

Στατιστικές Τεχνικές> Coding: Semi-Static Huffman: Byte-Oriented Huffman Code

- Huffman tree with degree 256 instead of 2
- Typically, the code assigned to each symbol contains between 1 and 5 bytes

Στατιστικές Τεχνικές> Coding: Semi-Static Huffman: Remarks

- Huffman coding allows perfoming **direct searching** on compressed text.
- The exact search can be done on the compressed text directly, using any known sequential pattern matching algorithm

Άλλοι τρόποι κωδικοποίησης

- Restricted Variable-Length Codes
  - Use first bit to indicate case.
  - 8 most frequent characters fit in 4 bits (0xxx).
  - 128 less frequent characters fit in 8 bits (1xxxxxxx)
  - In English, 7 most frequent characters are 65% of occurrences
  - Expected code length is approximately 5.4 bits per character, for a 32.8% compression ratio.
- Restricted Var-Length: Generalization for More Symbols
  - Use more than 2 cases.
  - 1xxx for $2^3 = 8$ most frequent symbols, and
  - 0xxx1xxx for next $2^6 = 64$ symbols, and
  - 0xxx0xxx1xxx for next $2^9 = 512$ symbols, and …
  - Average code length ~ 6.2 bits per symbol (23.0%) compression ratio.
  - Pro: Variable number of symbols. Con: Only 72 symbols in 1 byte.

Dictionary Methods

Dictionary Methods

- They achieve compression by replacing groups of consecutive symbols (or phrases) with a *pointer* to an entry in a **dictionary**
- Thus, the central decision in the design of a dictionary method is the selection of entries in the dictionary.
- The choice of phrases can be made by
  - static,
  - semi-adaptive, or
  - adaptive algorithms

Dictionary Methods> **Static Dictionaries**

- The simplest dictionary schemes use static dictionaries containing short phrases
- Example: Digram Coding
  - Idea: selected pairs of letters are replaced with codewords
  - at each step the next two characters are inspected and verified if they correspond to a digram in the dictionary
  - If so, they are coded together and the coding position is shifted by two characters; otherwise, the single character is represented by its normal code and the coding position is shifted by one character
- Weaknesses:
  - The dictionary may be suitable for one text and unsuitable for another.

Dictionary Methods>
Semi-Static and **Adaptive** Dictionaries

- Construct a new dictionary for each text to be compressed
- The problem of deciding which phrases to put in the dictionary is not an easy task
- Adaptive Dictionaries (Ziv-Lempel)
  - Idea: Replace strings of characters with a reference to a previous occurrence of the string.
  - This approach is effective because most characters can be coded as part of a string that has occurred earlier in the text
  - If the pointer to an earlier occurrence of a string is stored in fewer bits than the string it replaces, then compression is achieved
- Disadvanteages of Adaptive Dictionaries
  - they do not allow decoding to start in the middle of the compressed file (so, direct access is not possible unless we decode the text from its beginning)

Adaptive Dictionary Methods>
Lempel-Ziv Compression Algorithms

- Use the text already encountered to build the dictionary.
  - If text follows Zipf's laws, a good dictionary is built.
  - No need to store dictionary; encoder and decoder each know how to build it on the fly.
- Some variants: LZ77, Gzip, LZ78, LZW, Unix *compress*
- Variants differ on:
  - how dictionary is built,
  - how pointers are represented (encoded), and
  - limitations on what pointers can refer to.

Adaptive Dictionary Methods> LZ77(LZ1)

Data is encoded as a sequence of tuples:

<Number of characters back, Length, Next character>

- Example:
  - String: abaababbbbbbbbbbba
  - Encoding: <0,0,a> <0,0,b> <2,1,a> <3,2,b> <1,10,a>
  - Encoding: <0,0,a>
  - String: a
  - Encoding: <0,0,a> <0,0,b>
  - String: ab
  - Encoding: <0,0,a> <0,0,b> <2,1,a>
  - String: abaa
  - Encoding: <0,0,a> <0,0,b> <2,1,a> <3,2,b>
  - String: abaabab
  - Encoding: <0,0,a> <0,0,b> <2,1,a> <3,2,b><1,10,a>
  - String: abaababbbbbbbbbbba

Adaptive Dictionary Methods> LZ77(LZ1)

- Optimizations:
  - Limit size of back-pointers, e.g., 8K (13 bits).
  - Restrict length of phrases, e.g., 15 characters (4 bits).
  - Variable-length encode pointers and length.
- Encoding data structures:
  - Trie, hash table, or binary search tree.
- Characteristics:
  - Very fast decoding.
  - Low memory overhead.
  - Decoder is sufficiently small to include in compressed data.
    - Self expanding archives, typically found on PCs.

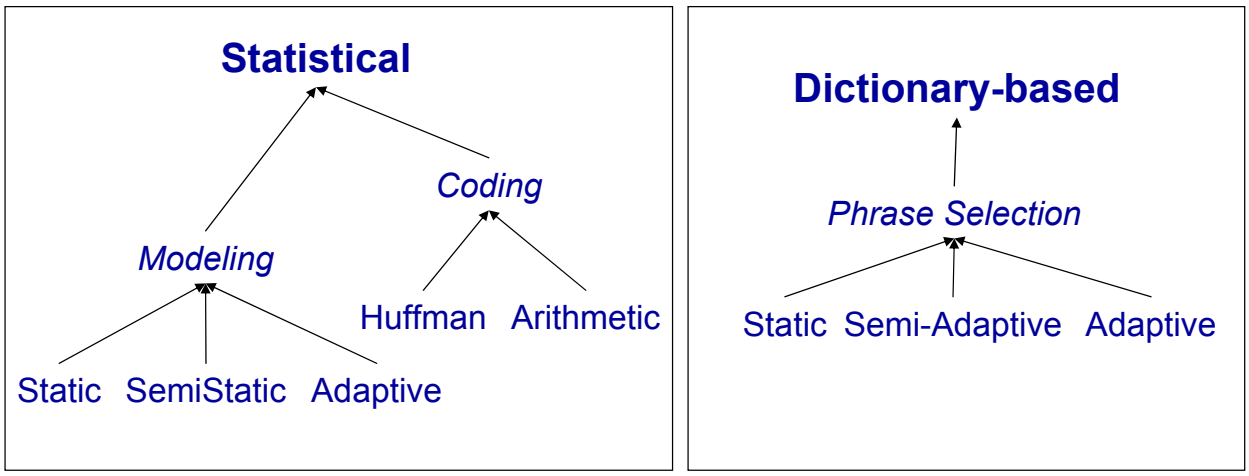
Adaptive Dictionary Methods> LZ77> Gzip

- **Gzip is a variant of LZ77**
  - Encoder locates previous strings using a hash table (three characters), then a linked list
  - User preferences (speed vs space) determine list length
  - For maximal compression, uses lookahead instead of simple greedy search for string matches
  - Uses one Huffman for offsets, another for lengths and characters
  - Huffman codes are semi-static:
    - Text is processed in chunks of up to 64K.
    - Each chunk has its own Huffman code.
    - Huffman codes are stored in the compressed text.

Adaptive Dictionary Methods> LZ78 (LZ2)

- **Data is encoded as a sequence of tuples:**
  - <Phrase ID, Next character>
  - Instead of looking backwards for substrings, use a phrase dictionary
- Phrase length does not need to be stored in the tuple.
- Phrase ids can take up less space than back pointers
- Phrase dictionary grows until a memory limit is reached.
- When full, dictionary:
  - is reinitialized,
  - is partially rebuilt, or
  - becomes static.
- Encodes faster than LZ77, decodes more slowly.

Compression Techniques: Summary



Παραδείγματα Τεχνικών Συμπίεσης

Generic File Compression

- files: gzip (LZ77), bzip (Burrows-Wheeler), BOA (PPM)
- archivers: ARC (LZW), PKZip (LZW+)
- file systems: NTFS

Communication

- Fax: ITU-T Group 3 (run-length + Huffman)
- Modems: V.42bis protocol (LZW) MNP5 (RL + Huffman)

Multimedia

- Images: gif (LZW), jbig (context), jpeg-ls (residual), jpeg (transform+RL+arithmetic)
- TV: HDTV (mpeg-4)
- Sound: mp3

Comparing Text Compression Techniques

	Arithmetic	Character Huffman	Word Huffman	Ziv-Lempel
Compression ratio	very good	poor	very good	good
Compression speed	slow	fast	fast	very fast
Decompression speed	slow	fast	very fast	very fast
Memory space	low	low	high	moderate
Random access	**no**	**yes**	**yes**	**no**