Text Operations

Berlin Chen 2003

References:

1. Modern Information Retrieval, chapters 7, 5
2. Information Retrieval: Data Structures & Algorithms, chapters 7, 8
3. Managing Gigabytes, chapter 2

Index Term Selection and Text Operations

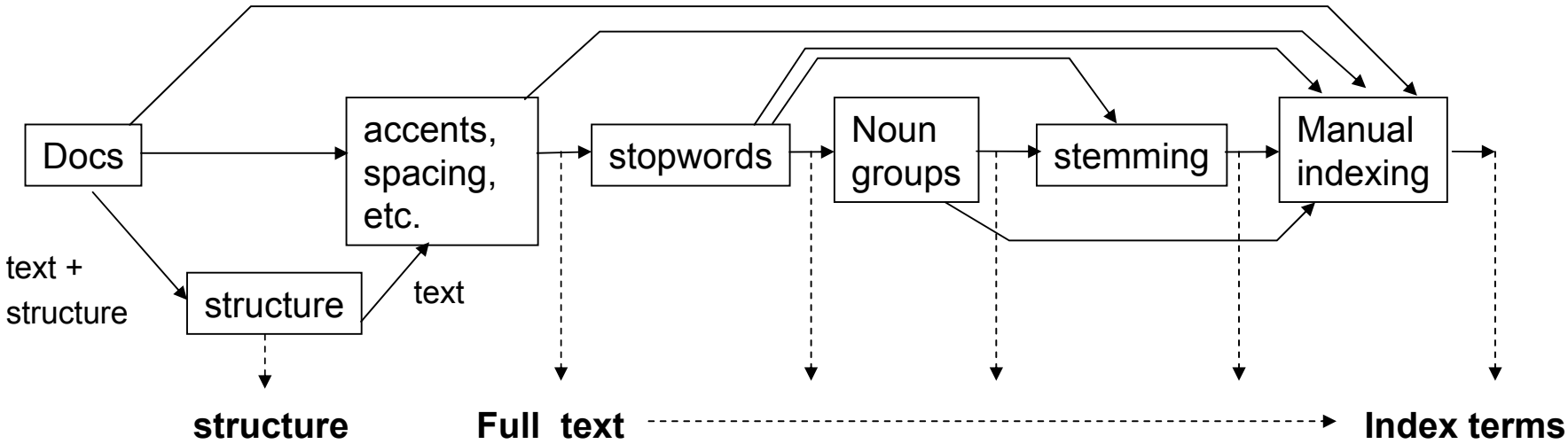
- Index Term Selection
  - Noun words (or group of noun words) are more representative of a doc content
  - Preprocess the text of docs in collection in order to select the meaningful/representative index terms
- Text Operations
  - During the preprocessing phrase, a few useful text operations can be performed
    - Lexical analysis
    - Eliminate of stop words
    - Stemming

    control the size of vocabulary (reduce the size of distinct index terms) side effect ?

    - Thesaurus construction/text clustering ⇨ improve performance but waste time
    - Text compressing ⇨ controversial for its benefits

Index Term Selection and Text Operations

- Logic view of a doc in text preprocessing



- Goals of Text Operations
  - Improve the quality of answer set
  - Reduce the space and search time

Document Preprocessing

- Lexical analysis of the text
- Elimination of stopwords
- Stemming the remaining words
- Selecting of indexing terms
- Construction term categorization structures
  - Thesauri
  - Word/Doc Clustering

Lexical Analysis of the Text

- Lexical Analysis
  - Convert a stream of characters (the text of document) into stream words or tokens
  - The major objectives is to identify the words in the text

- Four particular cases should be considered with care
  - Digits
  - Hyphens
  - Punctuation marks
  - The case of letters

Lexical Analysis of the Text

- **Numbers/Digits**
  - Most numbers are usually not good index terms
  - Without a surrounding context, they are inherently vague
  - The preliminary approach is to remove all words containing sequences of digits unless specified otherwise
  - The advanced approach is to perform date and number normalization to unify format
- **Hyphens**
  - Breaking up hyphenated words seems to be useful
  - But, some words include hyphens as an integrated part

Lexical Analysis of the Text

- **Punctuation marks**
  - Removed entirely in the process of lexical analysis
  - But, some are an integrated part of the word
- **The case of letters**
  - Not important for the identification of index terms
  - Converted all the text to either to either lower or upper cases
  - But, parts of semantics will be lost due to case conversion

The side effect of lexical analysis

User find it difficult to understand what the indexing strategy is doing at doc retrieval time.

Elimination of Stopwords

- **Stopwords**
  - Word which are too frequent among the docs in the collection are not good discriminators
  - A word occurring in 80% of the docs in the collection is useless for purposes of retrieval
    - E.g, articles, prepositions, conjunctions, …
  - Filtering out stopwords achieves a compression of 40% size of the indexing structure
  - **The extreme approach**: some verbs, adverbs, and adjectives could be treated as stopwords
- **The stopword list**

If queries are:

state of the art, to be or not to be, ….

Stemming

- Stem
  - The portion of a word which is left after the removal of affixes (prefixes and suffixes)
  - E.g., *V*(*connect*)={*connected, connecting, connection, connections, …* }
- Stemming
  - The substitution of the words with their respective stems
  - Methods
    - Affix removal
    - Table lookup
    - Successor variety (determining the morpheme boundary)
    - *N*-gram stemming based on letters' bigram and trigram information

Stemming: Affix Removal

- Use a suffix list for suffix stripping
  - E.g., **The Porter algorithm**
  - Apply a series of rules to the suffixes of words
    - Convert **plural forms** into **singular forms**
      - Words end in "*sses*"

        $sses \rightarrow ss$

        stresses → stress
      - Words end in "*ies*" but not "*eies*" or "*aies*"

        $ies \rightarrow y$
      - Words end in "*es*" but not "*aes*", "*ees*" or "*oes*"

        $es \rightarrow e$
      - Word end in "*s*" but not "*us*" or "*ss*"

        $s \rightarrow \phi$

Stemming: Table Lookup

- Store a table of all index terms and their stems

Term	Stem
engineering	engineer
engineered	engineer
engineer	engineer

  - Problems
    - Many terms found in databases would not be represented
    - Storage overhead for such a table

Stemming: Successor Variety

- Based on work in structural linguistics
  - Determine word and morpheme boundaries based on distribution of phonemes in a large body of utterances
  - The successor variety of substrings of a term will decrease as more characters are add until a segment boundary is reached
    - At this point, the successor will sharply increase
    - Such information can be used to identify stems

Prefix	Successor Variety	Stem
R	3	E, I,O
RE	2	A, D
REA	1	D
READ	3	A, I, S
READA	1	B
READAB	1	L
READABL	1	E
READABLE	1	BLANK

Stemming: N-gram Stemmer

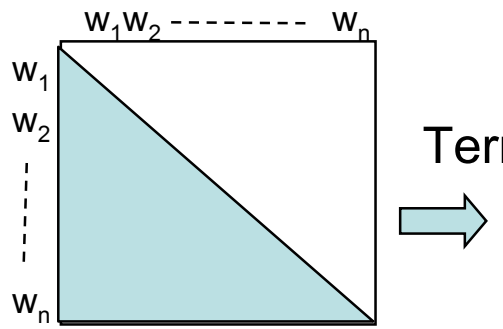
- Association measures are calculated between pairs of terms based on shared unique diagrams
  - diagram: or called the bigram, is a pair of consecutive letters
  - E.g.

    statistics → st ta at ti is st ti ic cs
    unique diagrams= at cs ic is st ta ti (7 unique ones)
    statistical → st ta at ti is st ti ic ca al
    unique diagrams= al at ca ic is st ta ti (8 unique ones)

    6 diagrams shared

  - Using Dice's coefficient

$$S = \frac{2C}{A+B} = \frac{2x6}{7+8} = 0.80$$



Term Clustering

Building a similarity matrix

Index Term Selection

- Full text representation of the text
  - All words in the text are index terms
- Alternative: an abstract view of documents
  - Not all words are used as index terms
  - A set of index terms (keywords) are selected
    - Manually by specialists
    - Automatically by computer programs
- Automatic Term Selection
  - **Noun words**: carry most of the semantics
  - **Compound words**: combine two or three nouns in a single component
  - **Word groups**: a set of noun words having a predefined distance in the text

Thesauri

- Definition of the thesaurus
  - A treasury of words consisting of
    - A precompiled list important words in a given domain of knowledge
    - A set of related words for each word in the list, derived from a synonymity relationship
  - More complex constituents (phrases) and structures (hierarchies) can be used
    - E.g., the Roget's thesaurus

cowardly *adjective* (膽怯的)
Ignobly lacking in courage: *cowardly turncoats*
Syns: chicken (slang), chicken-hearted, craven, dastardly, faint-hearted, gutless, lily-livered, pusillanimous, unmanly, yellow (slang), yellow-bellied (slang)

Thesauri: Term Relationships

form a hierarchical structure

automatically or by specialists

- Relative Terms (RT)
  - Synonyms and near-synonyms
    - Thesauri are most composed of them
  - Co-occurring terms — Depend on specific context
    - Relationships induced by patterns of within docs
- Broader Relative Terms (BT)
  - Like hypernyms (上義詞)
  - A word with a more general sense, e.g., animal is a hypernym of cat
- Narrower Relative Terms (NT)
  - Like hyponyms (下義詞)
  - A word with more specialized meaning, e.g., mare is a hyponym of horse

Thesauri: Term Relationships

Figure 1 shows an example of a poset representing geographic locations and sub-locations using a tree structure to show the partial ordering relation.

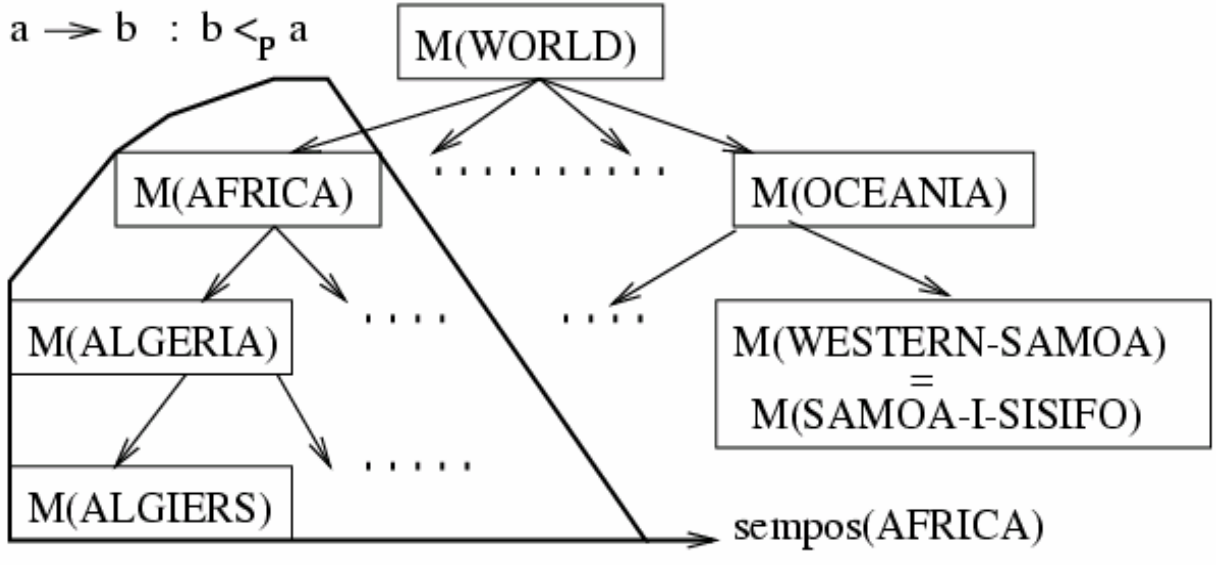


Figure 1: Example of Geographic Semantic Poset

Thesauri: Purposes

Forskett, 1997

- Provide a standard vocabulary (system for references) for indexing and searching
- Assist users with locating terms for proper query formulation
- Provide classified hierarchies that allow the broadening and narrowing of the current query request according to the needs of the user

Thesauri: Use in IR

- Help with the query formulation process
  - The initial query terms may be erroneous or improper
  - Reformulate the query by further including related terms to it
  - Use a **thesaurus** for assisting the user with the search for related terms

- **Problems**
  - **Local context** (the retrieved doc collection) vs. **global context** (the whole doc collection)
  - Time consuming

Text Compression

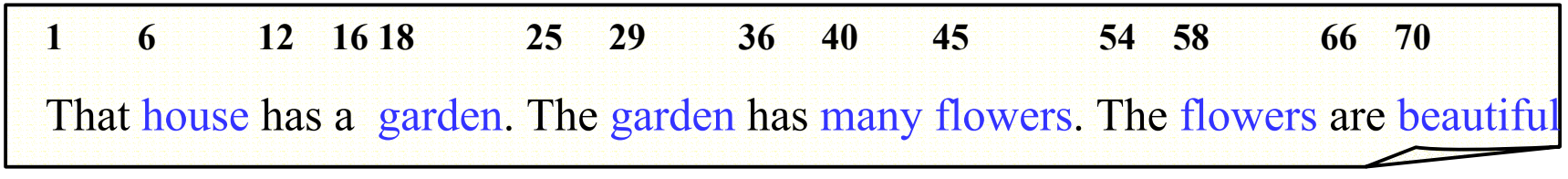
- Goals
  - Represent the text in fewer bits or bytes
  - Compression is achieved by identifying and using structures that exist in the text
  - The original text can be reconstructed exactly
    - text compression vs. data compression
- Features
  - **The costs reduced** is the space requirements, I/O overhead, and communication delays for digital libraries, doc databases, and the Web information
  - **The prices paid** is the time necessary to code and decode the text
    - How to randomly access the compressed text

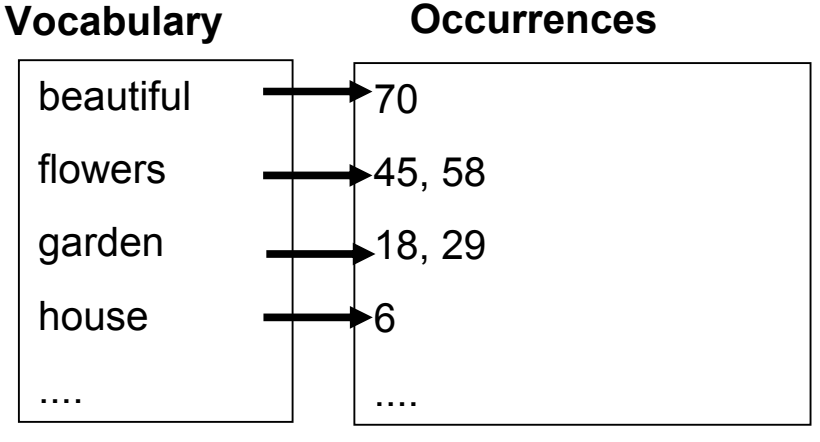
Text Compression

- **Considerations for IR systems**
  - The symbols to be compressed are words not characters
    - Words are **atoms** for most IR systems
    - Also better compression achieved by taking words as symbols
  - Compressed text pattern matching
    - Perform pattern matching in the compressed text without decompressed it
  - Also, compression for **inverted files** is preferable
    - Efficient index compression schemes

Text Compression: Inverted Files

- An inverted file is typically composed of
  - A vector containing all the distinct words (call vocabulary) in the text collection
  - For each vocabulary word, a list of all docs (identified by **doc number** in ascending order) in which that word occurs





An inverted list

Each element in a list points to a text position

An inverted file

Each element in a list points to a doc number

Text Compression: Basic Concepts

- Two general approaches to text compression
  - Statistical (symbolwise) methods
  - Dictionary methods

- Statistical (symbolwise) methods
  - Rely on generating good probability estimates for each symbol in the text
  - A symbol could be a character, a text words, or a fixed number of characters
  - **Modeling**: estimates the probability on each next symbol, forms a collection of probability distributions
  - **Coding**: converts symbols into binary digits
  - **Strategies**: Huffman coding or Arithmetic coding

Text Compression: Basic Concepts

- Statistical methods (cont.)
  - Hoffman coding
    - Each symbols is pre-coded using a fixed number of bits
    - Compression is achieved by assigning a small number of bits to symbols with higher probabilities
    - Coder and decoder refer to the same model
  - Arithmetic coding
    - Compute the code incrementally one symbol at a time
    - **Does not allow random access** to the compressed files

Text Compression: Basic Concepts

- Dictionary methods
  - Substitute a sequence of symbols by a pointer to a previous occurrence sequence
  - The pointer representations are references to entries in a dictionary composed of a list of symbols (phrases)
  - Methods: Ziv-Lempel family

- **Compression ratios** for English text
  - Character-based Huffman: 5 bits/character
  - Word-based Huffman: over 2 bits/character (20% ↑)
  - Ziv-Lempel: lower 4 bits/character
  - Arithmetic: over 2 bits/character

Statistical Methods

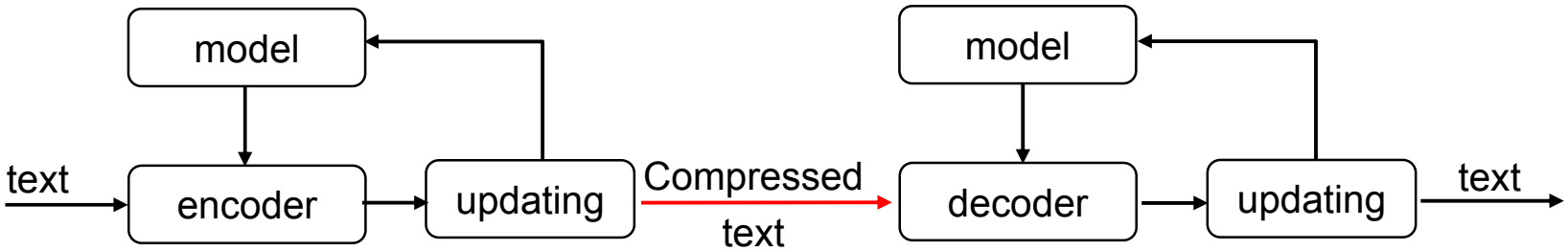
- Three Kinds of Compression Models
  - **Adaptive Modeling**
    - Start with no information about the text
    - Progressively learn about its statistical distribution as the compression process goes on
    - *Disadvantage*: can't not provide random access to the compressed file
  - **Static Modeling**
    - The distribution for all input text is known beforehand
    - **Use the same model** (**probability distribution**) perform **one-pass compression** regardless of what text is being coded
    - *Disadvantage*: probability distribution deviation

Statistical Methods

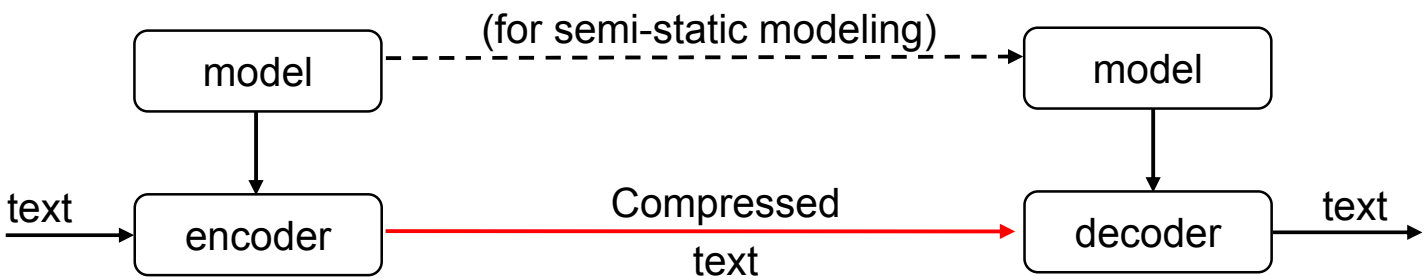
- Three Kinds of Compression Models (cont.)
  - **Semi-static modeling**
    - Do not assume any distribution of the data but learn it in the first pass
      - Generate a model specifically for each file that is to be compressed
    - In the second pass, the compression process goes on based on the estimates
    - Disadvantages
      - Two-pass processing
      - The probability distribution should be transmitted to the decoder before transmitting the encode data

Statistical Methods

- Using a Model to Compress Text
  - Adaptive modeling



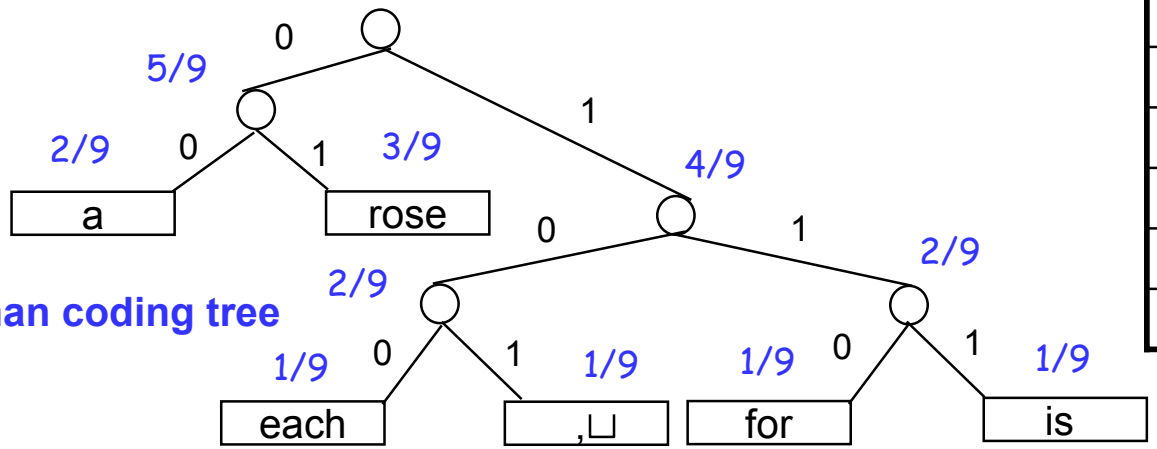
  - Static/Semi-static modeling



Statistical Methods: Huffman Coding

- Ideas
  - Assign a variable-length encoding in bits to each symbol and encode and encode each symbol in turn
  - Compression achieved by assigned shorter codes to more frequent symbols
  - **Uniqueness**: No code is a prefix of another

Original text: for each rose, a rose is a rose



Huffman coding tree

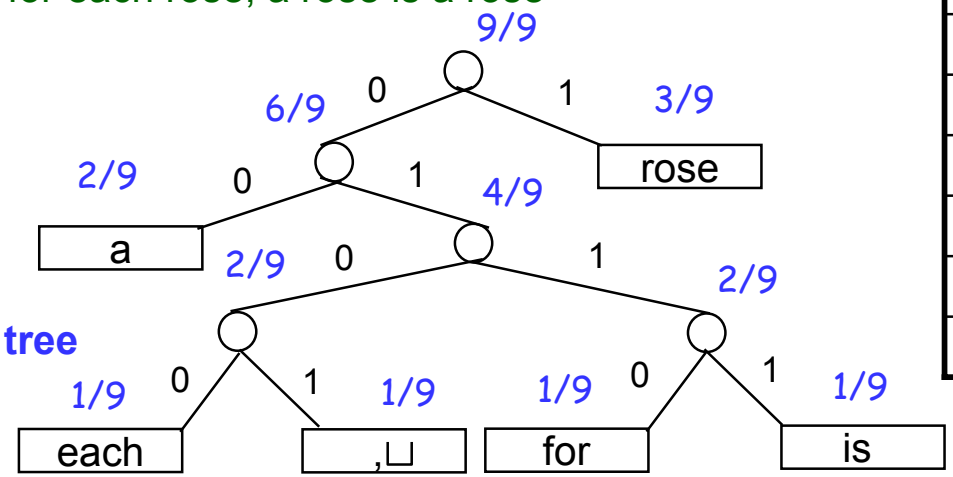
Symbol	Prob.	Code
each	1/9	100
,␣	1/9	101
for	1/9	110
is	1/9	111
a	2/9	00
rose	3/9	01

Average=2.44 bits/per sample

Statistical Methods: Huffman Coding

- But in the figure of textbook (???)

Original text: for each rose, a rose is a rose



Huffman coding tree

Symbol	Prob.	Code
each	1/9	0100
,␣	1/9	0101
for	1/9	0110
is	1/9	0111
a	2/9	00
rose	3/9	1

Average=2.56 bits/per sample

$$E = \sum - p_i \log_2 p_i$$
$$= -(4 \times \frac{1}{9} \times \log_2 \frac{1}{9} + \frac{2}{9} \times \log_2 \frac{2}{9} + \frac{3}{9} \times \log_2 \frac{3}{9})$$
$$\approx 2.42$$

Statistical Methods: Huffman Coding

- Algorithm: **an bottom-up approach**
  - First, a forest of one-node trees (each for a distinct symbol) whose probabilities sum up to 1
  - Next, two nodes with the smallest probabilities become children of a new created parent node
    - The probability of the parent node equals to the sum of the probabilities of two children nodes
    - **Nodes that are already children are ignored** in the following process
  - Repeat until only one root node of the decoding tree is formed

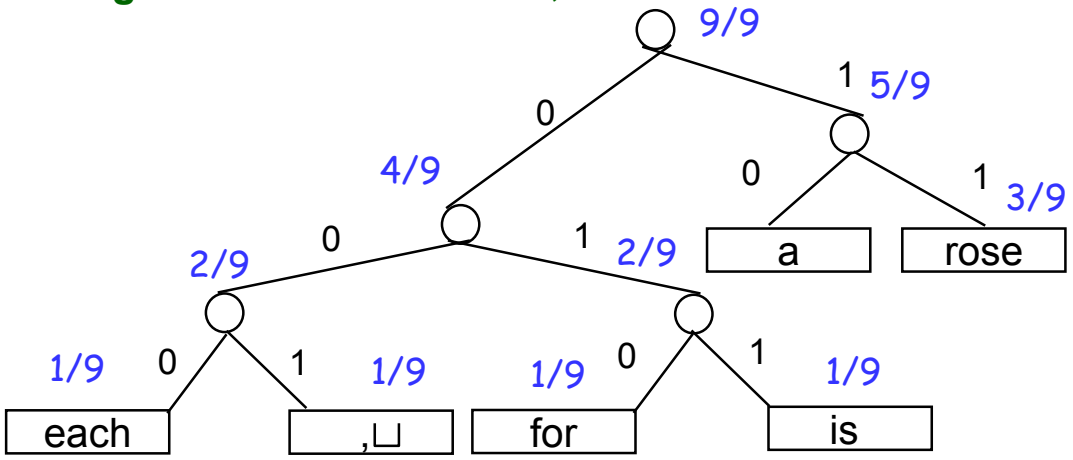
The number of trees finally formed will be quite large!

- The interchanges of the left and right subtrees of any internal node

Statistical Methods: Huffman Coding

- The **canonical** tree
  - The height of the left subtree of any node is never smaller than that of the right subtree
  - All leaves are in increasing order of probabilities from left to right
  - **Property**: the set of code with the same length are the binary representations of consecutive integers

Original text: for each rose, a rose is a rose



canonical Huffman coding tree

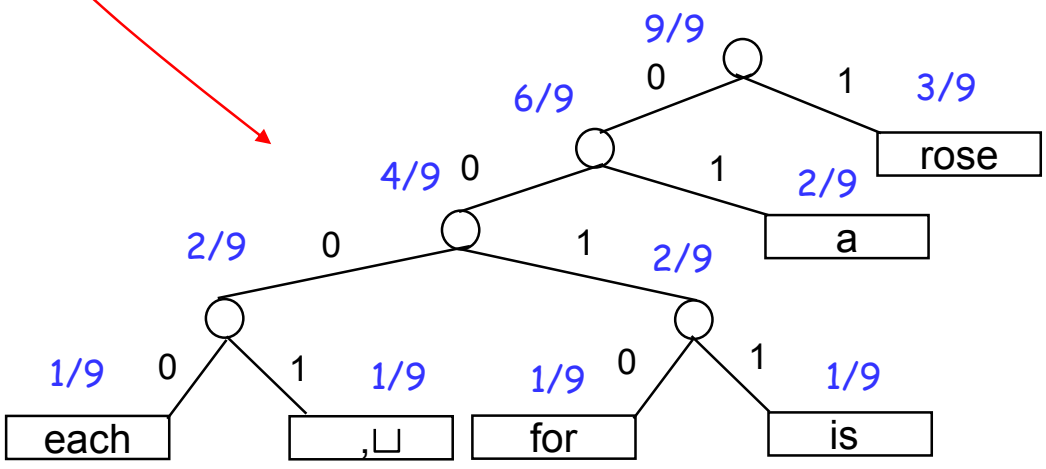
Symbol	Prob.	Old Code	Can. Code
each	1/9	100	000
,␣	1/9	101	001
for	1/9	110	010
is	1/9	111	011
a	2/9	00	10
rose	3/9	01	11

Average=2.44 bits/per sample

Statistical Methods: Huffman Coding

- The **canonical** tree
  - But in the figure of textbook (???)

Original text: for each rose, a rose is a rose



canonical Huffman coding tree

Symbol	Prob.	Old Code	Can. Code
each	1/9	0100	0000
,␣	1/9	0101	0001
for	1/9	0110	0010
is	1/9	0111	0011
a	2/9	00	01
rose	3/9	1	1

Average=2.56 bits/per sample

$$E = \sum - p_i \log_2 p_i$$
$$= -(4 \times \frac{1}{9} \times \log_2 \frac{1}{9} + \frac{2}{9} \times \log_2 \frac{2}{9} + \frac{3}{9} \times \log_2 \frac{3}{9})$$
$$\approx 2.42$$

Dictionary Methods: Ziv-Lempel coding

- Idea:
  - Replace strings of characters with a reference to a previous occurrence of the string
- Features:
  - Adaptive and effective
  - Most characters can be coded as part of a string that has occurred earlier in the text
  - Compression is achieved if the reference, or pointer, is stored in few bits than the string it replaces
- Disadvantage
  - Do not allow decoding to start in the middle of a compressed file (direct access is not possible)

Comparison of the 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
Compressed pattern matching	no	yes	yes	Yes (theoretically)
Random access	no	yes	yes	no

- "very good": compression ratio under 30%
- "good": compression ratio between 30% and 45%
- "poor": compression ratio over 45%

Inverted File Compression

- An Inverted File composed of
  - Vocabulary
  - Occurrences (lists of ascending doc numbers or word positions)
- The lists can be compressed
  - E.g., considered as a sequence of gaps between doc numbers
    - IR processing is usually done starting from the beginning of the lists
    - Original doc numbers can be recomputed through sums of gaps
    - Encode the gaps: smaller ones (for frequent words) have shorter codes

Trends and Research Issues

- **Text Preprocessing** for indexing
  - Lexical analysis
  - Elimination of stop words
  - Stemming
  - Selection of indexing terms
- **Text processing** for query reformulation
  - Thesauri (term hierarchies or relationships)
  - Clustering techniques
- **Text compression** to reduce space, I/O, communication costs
  - Statistical methods
  - Dictionary methods