

Indexing Basics

Part-1

Text Collections and IR

- Large collections of documents from various sources: news articles, research papers, books, digital libraries, Web pages, etc.

Sample Statistics of Text Collections

- Dialog:
 - claims to have more than 15 terabytes of data in >600 Databases, > 800 million unique records
- LEXIS/NEXIS:
 - claims 7 terabytes, 1.7 billion documents, 1.5 million subscribers, 11,400 databases; >200,000 searches per day; 9 mainframes, 300 Unix servers, 200 NT servers
- Web Search Engines:
 - Google claim to index over 1.5 billion pages.
 - How many search engines are available these days?
- TREC collections:
 - total of about 5 gigabytes of text

Designing an IR System

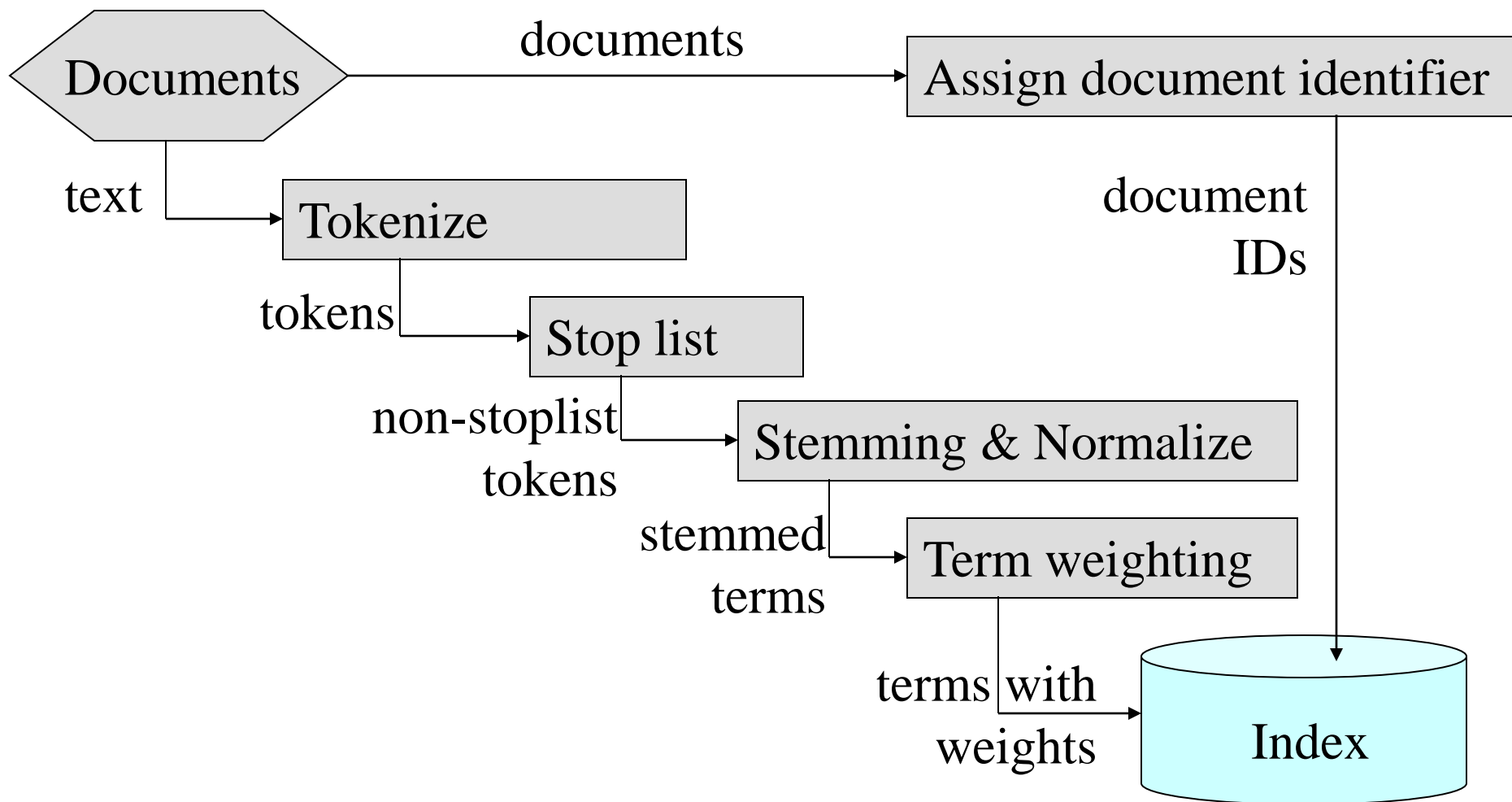
Our focus during IR system design is:

- In improving performance effectiveness of the system
 - Effectiveness of the system is measured in terms of precision, recall, ...
 - Stemming, stopwords, weighting schemes, matching algorithms
- In improving performance efficiency.

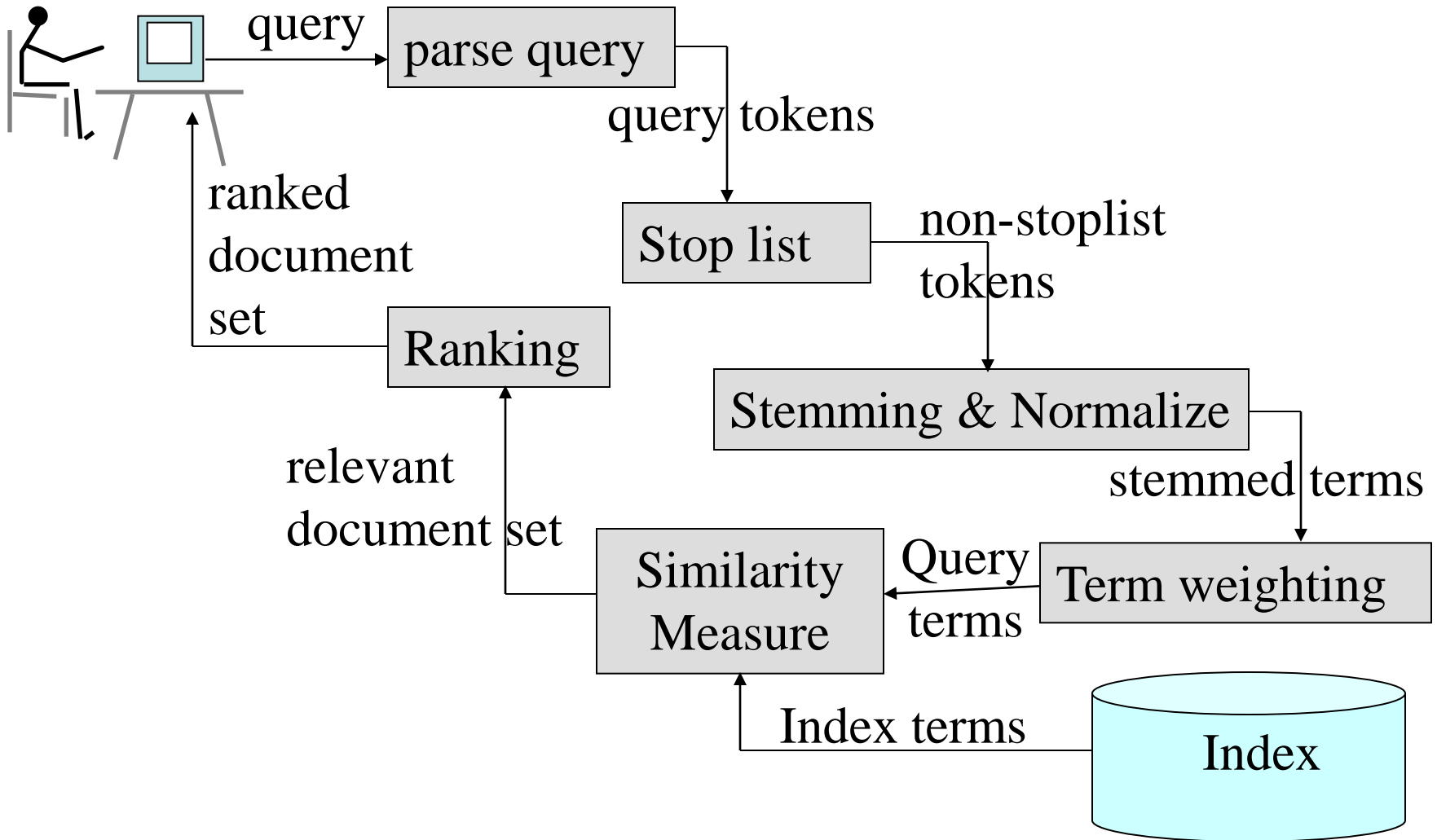
The concern here is

- storage space usage, access time, ...
- Compression, data/file structures, space – time tradeoffs
- The two subsystems of an IR system:
 - Searching and
 - Indexing

Indexing Subsystem



Searching Subsystem



Basic assertion

Indexing and searching:

inexorably connected

- you cannot search that that was not first indexed in some manner or other
- indexing of documents or objects is done in order to be searchable
 - there are many ways to do indexing
- to index one needs an indexing language
 - there are many indexing languages
 - even taking every word in a document is an indexing language

Knowing searching is knowing indexing

Implementation Issues

- **Storage of text:**

- The need for text **compression: to** reduce storage space

- **Indexing text**

- Organizing indexes
 - What techniques to use ? How to select it ?
 - Storage of indexes
 - Is compression required? Do we store on memory or in a disk ?

- **Accessing text**

- Accessing indexes
 - How to access to indexes ? What data/file structure to use?
 - Processing indexes
 - How to search a given query in the index? How to update the index?
 - Accessing documents

Text Compression

- Text compression is about finding ways to represent the text in fewer bits or bytes

- **Advantages:**

- save storage space requirement.
- speed up document transmission time
- Takes less time to search the compressed text

Disadvantage

The time required to decode and encode the text.

- **Common compression methods**

- Static methods:** which requires statistical information about frequency of occurrence of symbols in the document

E.g. Huffman coding

- Estimate probabilities of symbols, code one at a time, shorter codes for high probabilities

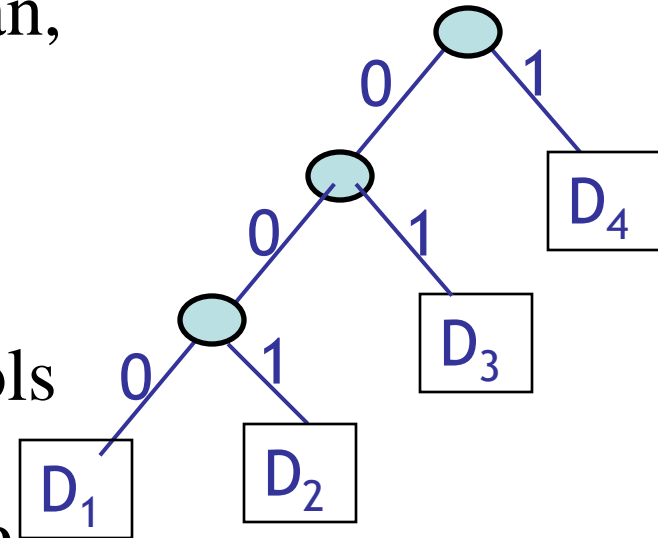
- Adaptive methods:** which constructs dictionary in the processing of compression

E.g. Ziv-Lempel compression:

- Replace words or symbols with a pointer to dictionary entries

Huffman coding

- Developed in 1950s by David Huffman, widely used for text compression, multimedia codec and message transmission
- **The problem:** Given a set of n symbols and their weights (or frequencies), construct a tree structure (a binary tree for binary code) with the objective of reducing memory space & decoding time per symbol.
- Huffman coding is constructed based on frequency of occurrence of letters in text documents



Code of:

$D_1 = 000$

$D_2 = 001$

$D_3 = 01$

$D_4 = 1$

How to construct Huffman coding

Step 1: Create forest of trees for each symbol, t_1, t_2, \dots, t_n

Step 2: Sort forest of trees according to falling probabilities of symbol occurrence

Step 3: WHILE more than one tree exist DO

- Merge two trees t_1 and t_2 with least probabilities p_1 and p_2
- Label their root with sum $p_1 + p_2$
- Associate binary code: 1 with the right branch and 0 with the left branch

Step 4: Create a unique code word for each symbol by traversing the tree from the root to the leaf.

- Concatenate all encountered 0s and 1s together during traversal
- The resulting tree has a prob. of 1 in its root and symbols in its leaf node.

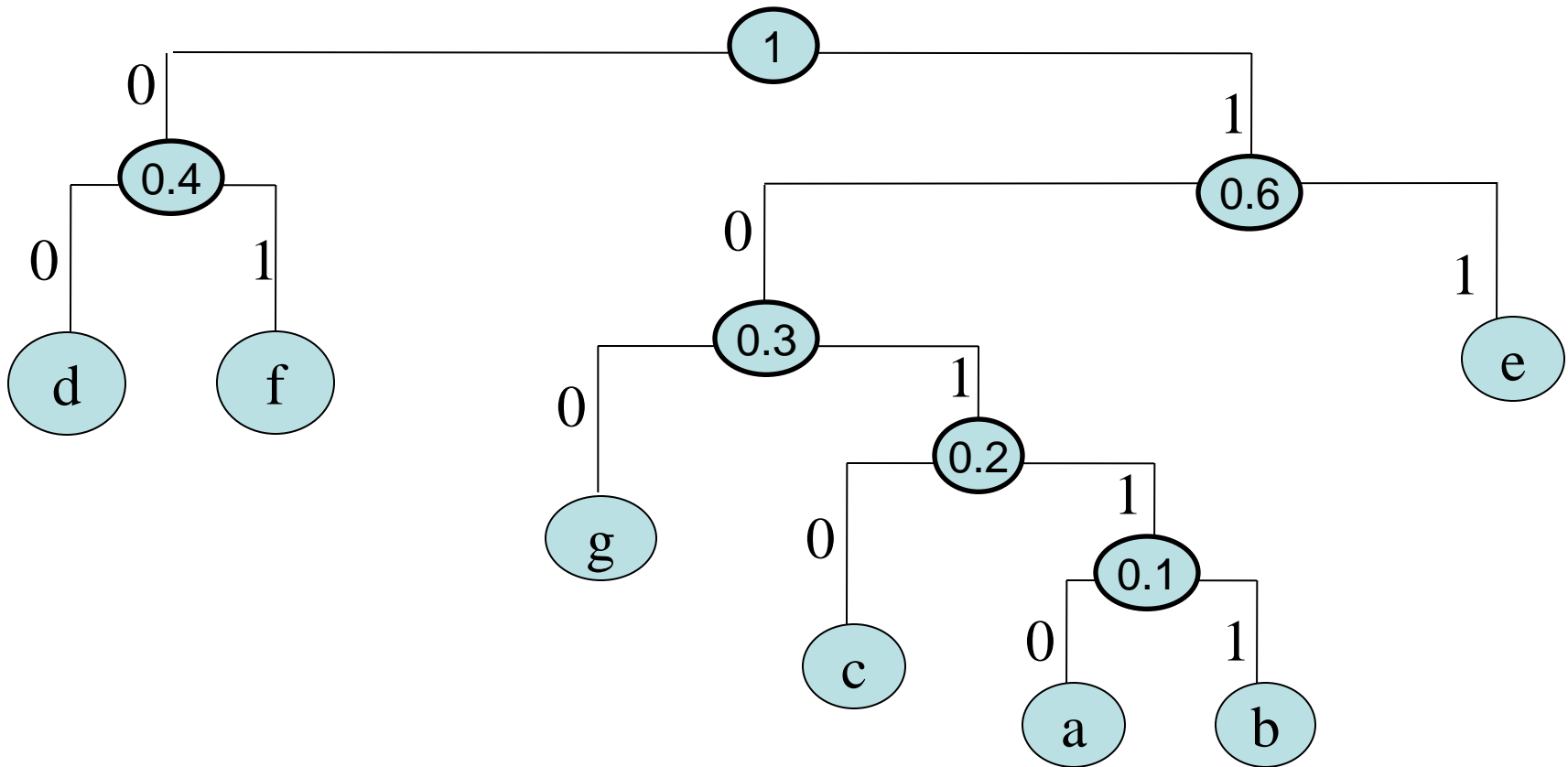
Example

- Consider a 7-symbol alphabet given in the following table to construct the Huffman coding.

Symbol	Probability
a	0.05
b	0.05
c	0.1
d	0.2
e	0.3
f	0.2
g	0.1

- The Huffman encoding algorithm picks each time two symbols (with the smallest frequency) to combine

Huffman code tree



- Using the Huffman coding a table can be constructed by working down the tree, left to right. This gives the binary equivalents for each symbol in terms of 1s and 0s.
- What is the Huffman binary representation for 'café'?

Exercise

1. Given the following, apply the Huffman algorithm to find an optimal binary code:

Character:	a	b	c	d	e	t
Frequency:	16	5	12	17	10	25

Ziv-Lempel compression

- The problem with Huffman coding is that it requires knowledge about the data before encoding takes place.
 - Huffman coding requires frequencies of symbol occurrence before codeword is assigned to symbols
- **Ziv-Lempel compression**
 - Not rely on previous knowledge about the data
 - Rather builds this knowledge in the course of data transmission/data storage
 - Ziv-Lempel algorithm (called LZ) uses a table of code-words created during data transmission;
 - each time it replaces strings of characters with a reference to a previous occurrence of the string.

Lempel-Ziv Compression Algorithm

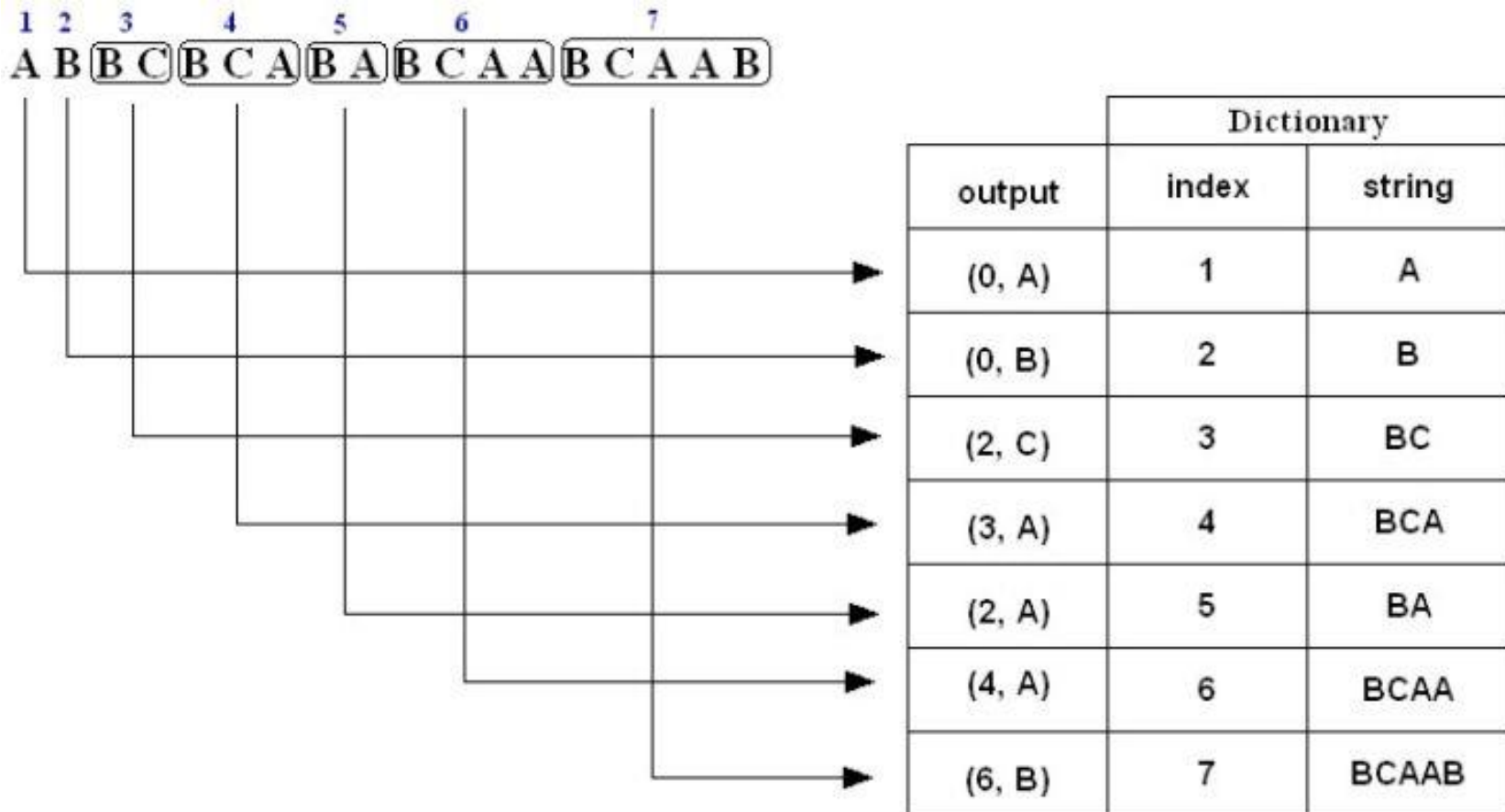
- The multi-symbol patterns are of the form: $C_0C_1 \dots C_{n-1}C_n$. The **prefix** of a pattern consists of all the pattern symbols except the last: $C_0C_1 \dots C_{n-1}$

Lempel-Ziv Output: there are three options in assigning a code to each symbol in the list

- If one-symbol pattern is not in dictionary, assign (0, symbol)
- If multi-symbol pattern is not in dictionary, assign (dictionaryPrefixIndex, lastPatternSymbol)
- If the last input symbol or the last pattern is in the dictionary, assign (dictionaryPrefixIndex,)

Example: LZ Compression

Encode (i.e., compress) the string **ABBCBCABABCAABCAAB** using the LZ algorithm.



The compressed message is: **(0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)**

Note: The above is just a representation, the commas and parentheses are not transmitted;

Example: Decompression

Decode (i.e., decompress) the sequence (0, A) (0, B) (2, C) (3, A) (2, A) (4, A) (6, B)

1 2 3 4 5 6 7
(0, A) (0, B) (2, C) (3, A) (2, A) (4, A) (6, B)

↓ ↓ ↓ ↓ ↓ ↓ ↓

A B string(2) string(3) string(2) string(4) string(6)
+ C + A + A + A + B

Dictionary		
output	index	string
A	1	A
B	2	B
BC	3	BC
BCA	4	BCA
BA	5	BA
BCAA	6	BCAA
BCAAB	7	BCAAB

The decompressed message is: •
ABBCBCABABCAABCAAB

Exercise

Encode (i.e., compress) the following strings using the Lempel-Ziv algorithm.

1. Mississippi
2. ABBCBCABABCAABCAAB
3. SATATASACITASA.

Indexing: Basic Concepts

- Indexing is used to speed up access to desired information from document collection as per users query such that
 - It enhances efficiency in terms of time for retrieval. Relevant documents are searched and retrieved quick

Example: author catalog in library

- An **index file** consists of records, called **index entries**.
- **Index files** are much smaller than the original file.
 - For Example: 1 GB of TREC text collection the vocabulary has a size of only 5 MB (Ref: Baeza-Yates and Ribeiro-Neto, 2005)
 - This size may be further reduced by **Linguistic pre-processing** (like stemming & other normalization methods).
- The usual unit for indexing is the word
 - **Index terms** - are used to look up records in a file.

How Current Search Engines index?

- Indexes are built using a **web crawler**, which retrieves each page on the Web for indexing.
 - After indexing, the local copy of each page is discarded, unless stored in a cache.
- Some search engines: automatically index
 - such search engines include:
Google, AltaVista, Excite, HotBot, InfoSeek, Lycos
- Some others: semi automatically index
 - Partially human indexed, hierarchically organized
 - Such search engines include:
Yahoo, Magellan, Galaxy, WWW Virtual Library
- Common features
 - allow Boolean searches

Search Engines

- Search: to examine a computer file, disk, database, or network for particular information.
- Engine: something that supplies the driving force or energy to a movement, system, or trend.
- Search Engine: a computer program that searches for particular keywords and returns a list of documents in which they were found, especially a commercial service that scans documents on the Internet

Cont'

- Search Engines are special sites on the web that are designed to help people find information stored on other sites.

They perform three basic tasks:

- Search the web,
- Keep an index of the words they find, and where they find them,
- Allow users to look for words or combination of them found in that index

Brief History of Search Engines

Yahoo!(www.yahoo.com) -(1994-) directory service and search engine.

- Infoseek** –(1994-2001) search engine.
- Inktomi** –(1995-) search engine infrastructure, acquired by Yahoo! 2003.
- AltaVista**–(1995-) search engine, acquired by Overture in 2003.
- AlltheWeb** –(1999-) search engine, acquired by Overture in 2003.
- Ask Jeeves**(www.ask.com) -(1996-) Q&A and search engine, acquired by IAC/InterActiveCorp in 2005.
- Overture** –(1997-) pay-per-click search engine, acquired by Yahoo! 2003.
- MSN Search** (www.msn.com) –(2004-) Microsoft Network's search engine.
- Google** (www.google.com)–(1998-) –search engine.

How Search Engines work?

- **Spiders:** a special software robot to build lists of words found on web sites,
- **Web Crawling:** the process of building lists by a spider,
 - How it starts searching the web:
 - Starts with heavily used and popular servers,
 - Indexing the words on the sites
 - Words occurring in the title, sub title, meta-tags, etc. are noted for special consideration during a subsequent user search,
- **Meta tags** allow the owner of the page to specify key words under which the page will be indexed.

Major Steps in Index Construction

- **Source file:** Collection of text document
 - A document can be described by a set of representative keywords called index terms.
- **Index Terms Selection:**
 - Tokenize:** identify words in a document, so that each document is represented by a list of keywords or attributes
 - Stop words:** removal of high frequency words
 - Stop list of words is used for comparing the input text
 - Word stem:** reduce words with similar meaning into their stem/root word
 - Suffix stripping is the common method
 - Term relevance weight:** Different index terms have varying relevance when used to describe document contents.
 - This effect is captured through the **assignment of numerical weights to each index term** of a document.
 - There are different index terms weighting methods: **TF**, **TF*IDF**, ...
- **Output:** a set of index terms (vocabulary) to be used for **Indexing** the documents that each term occurs in.

Basic Indexing Process

*Documents to
be indexed.*



Friends, Roman countrymen.

*Token
stream.*

Tokenizer

Friends Romans countrymen

*Modified
tokens.*

**Linguistic
preprocessing**

friend roman countryman

*Index File
(Inverted file).*

Indexer

friend

roman

countryman



Building Index file

- An index file of a document is a file consisting of a list of index terms and a link to one or more documents that has the index term
 - A good **index file** maps each keyword K_i to a set of documents D_i that contain the keyword



- Index file usually has index terms in a sorted order.
 - The sort order of the terms in the index file provides an order on a physical file
- An index file is list of search terms that are organized for associative look-up, i.e., to answer user's query:
 - In which documents does a specified search term appear?
 - Where within each document does each term appear? (There may be several occurrences.)
- For organizing index file for a collection of documents, there are various options available:
 - Decide what data structure and/or file structure to use. Is it sequential file, inverted file, suffix array, signature file, etc. ?

Index file Evaluation Metrics

- Running time
 - Access/search time
 - Update time (Insertion time, Deletion time,)
- Space overhead
 - Computer storage space consumed.
- Access types supported efficiently. Is the indexing structure allows to access:
 - records with a specified term, or
 - records with terms falling in a specified range of values.

END