Indexing

- Indexes are used to increase the performance of IR systems

- Text indexing is the process of deciding what will be used to represent a document
Indexes

Index is accessed by the atoms of a query language. These atoms are terms, keys, or features. Most common terms are:

- words in text
- manually assigned terms
- documents structures (sentences, paragraph)
- document links
- term sequences (phrases, names, dates)
- term sets (thesaurus or synonym classes)
- concepts
Basic Indexing Processes

- Parse documents to recognize structures such as title, abstract, section
- Scan for word tokens (lexical analysis)
- Remove stopwords
- Stem words
- Select index terms
  - nouns
  - phrases
  - concepts (thesaurus classes)
- Weight terms
Removal of Stopwords

- Remove a list of common words such as articles, prepositions, and conjunctions

- Save storage

- Verbs, adverbs, adjectives could be stopwords too.

- Removal of stopwords can reduce recall.
Stemming

- Stemming is commonly used to conflate morphological variants
- Typical stemming systems use a set rules and/or dictionaries, for example, s suffix rule
- Small improvement in effectiveness
- Without domain knowledge, it may lead to retrieval occasional failures
- Miss some good conflations
- Too aggressive in stemming (e.g., police/policy)
Phrases

• Phrases can have an important impact on both efficiency and effectiveness

• Both statistical and syntactical methods may be used to identify phrases
  – speed up phrase queries
  – increase precision (e.g., black sea, not black and sea)

• Find word pairs occurred more than a given number of times in all documents.
Thesaurus

- Use a thesaurus database such as Wordnet. It may not be effective because it is not domain specific.

- Automatic generation of thesaurus based on term co-occurrence.

- Use thesaurus in query expansion.
Encode And Compression

• Encoding: transforms text from one representation to another representation

• Compression is an encoding for less space

• Lostless compression: decoder can reproduce the message exactly

• Lossy compression: can reproduce the message approximately
Advantages of Compressions

- Save memory space
- Save secondary storage space
- Save accessing time
- Save transmission time
Disadvantages of Compression

- encoding and decoding time
- complicates or prevents random access
- makes data corruption much more costly
Compression Methods

• Fixed Length
  – short bytes

• Restrict Variable Length
  – Basic method

• Variable Length
  – Huffman code
  – Lempel Ziv (Gzip and Unix compress)
Compression Methods

- Character Based
  - Encoding individual characters

- Word Based
  - Encoding words
Huffman Coding

- each symbol is assigned a variable-length encoding in bits
- frequent symbols are assigned fewer bits
- infrequent symbols are assigned more bits
- no code is a prefix of another.
Huffman Coding

• Gather probabilities for symbols
  – characters or words
• Build a binary tree as follows:
  – get the two least frequent symbols/nodes, form a parent node
  – label the least probable branch 0, label the other 1
  – \[ P(\text{node}) = P(\text{child}_1) + P(\text{child}_2) \]
  – continues until there is only one node left
Huffman Coding

- the path from the root to the leaf is the code of the symbol
- frequent symbols are near the root, giving them shorter codes
- less frequent symbols are deeper, giving them longer codes
Huffman Coding

- there are many different encodings (interchanging left and right subtrees)
- many codes are not assigned to any symbols
- English text, with symbols for characters is approximately 5 bits per character (37.5% compression rate)
- need a bit-by-bit scan of stream for decoding
- it’s inefficient to look up codes. The entire tree needs to be stored.
- Symbols could be words and better compression has been achieved
Cannonical Huffman Codes

- a Huffman tree is cannonical if its height of left subtree is not less than that of its right subtree, and all leaves are in increasing order of probabilities from left to right.
- the codes with the same length are consecutive integers.
- it is not necessary to store the entire tree. A table of contents of containing the first entry of each code length is enough.
- decoding is more efficient.
Byte-Oriented Huffman Code

- each edge is labeled as a byte and each node has up to 256 children.
- empty nodes should be in leaf nodes.
- compression rate is similar to that of bit Huffman code
- compression and decompression are much faster because of faster byte processing
- direct searching on compressed text is possible