Document Properties and Processing

Parsing and Linguistic Analysis

Chapter 4 and 6

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2015 ; distribution outside this group of students is NOT allowed.*

When a file is returned to crawler central (and say stored in the Repository of web-based documents), the indexer retrieves this document and tries to determine whether it is indexable or not based on the suffix of the file-name retrieved. If the suffix is not sufficient in figuring out the exact type of the document (or there is no suffix), then some type of parsing needs to be pursued to determine the indexability of the document and its contents (do they include text or not). This is what we also call `structure recognition`. During this phase other important pieces of information are collected about the document/file in question. These include the character set used (ASCII vs UNICODE for example), possibly the language used (in ASCII a code-page's upper code usage might be indicative of it, in UNICODE the 2-byte characters of a foreign language is an easier way to recognize the language) for the text, and also the programming language that might be used in the file (e.g. several UNIX script files begin with a declaration `#!/usr/local/bin/perl` to indicate that the text file is a Perl-script, or `#!/usr/local/bin/python` to indicate a Python script).

At the same time, the tokenization component needs to be initiated and for that several data structures need to be initialized. Such initializations might depend on the programming language or in general, the language used, the character set used, and the type of the file name (since a parser for a specific programming language might have to be used to reliably parse the file).

For an English text, or English-based texts, or in general documents expressing a spoken language several pieces of information are known in advance (a-priori).

Approximating Web document size: Pareto distribution

What can we say about the distribution of document sizes in the Web? How can we approximate the distribution of size for Web-based document? How can for example the crawler anticipate the amount of space it will use for fetching information, or the indexer to plan ahead of time for the 1 billion Web-based documents that it plant to index? Web-based document size distribution changes between text and image documents and thus depends on the document type. In any case, the size of a document can be approximated by a Pareto distribution.

Pareto distribution. Web-document size can be approximated to follow a **Pareto distribution** with probability density function

$$p(x) = \theta k^{\theta}/x^{1+\theta}, \text{and thus } Pr(X \geq x) \approx \left(\frac{k}{x}\right)^{\theta} \text{ i.e. } Pr(X \leq x) \approx 1 - \left(\frac{k}{x}\right)^{\theta}.$$

where x is measured in bytes and $k, \theta > 0$ are parameters of the distribution. ($p(x)$ gives the probability that a document is of size x, and $Pr(X \geq x)$ is the probability that the size X of a document is at least x bytes long.)

Between the two parameters k and θ, parameter θ might change very slowly with time. On the other hand parameter k might grow significantly if say, video/audio files become more frequent and thus contribute to the increase of file size.

Choosing θ. For text files, a $\theta = 1.36$ is used and smaller values can be used for image files and other binary formats. A typical generic choice for all file types is $\theta = 1.1$.

Choosing k. The choice of k varies with time. For example $k = 9.3KB, 18.7KB, 130KB, 400KB$ is a best fit for 1998, 2003, 2009, and 2013 respectively.

Example. For $k = 9.3KB$, and $\theta = 1.1$, if we use the expression for $Pr(X \geq x)$ we conclude that 93% of the documents have size no more that 9.3KB.

Power Law. Note that the Pareto density function is of the form c/x^{θ}, for some constant c. Such distributions are said to follow the power law. The degree of a vertex of the Web graph also seems to follow a power law distribution with $\theta = 2.1 - 2.5$.

Character-based entropy. The amount of information in a document is known as its **entropy**. Let a text document have N characters. Out of this document of file length N, there are n distinct characters of frequency f_i for character i, where $i = 1, \ldots, n$. Thus $\sum_{i=1}^{n} f_i = N$. Then the "probability" of occurrence of character i in the text is denoted by $p_i = f_i/N$ (or equivalently, this is the fraction of the text using character i). Then, the entropy of the document is

$$E = -\sum_{i=1}^{n} p_i \lg(p_i),$$

where n is the number of different characters and $\sum_i f_i = N$ is the length/size of the document.

Example 1 (binary alphabet). Consider a document with N zeroes and ones (the only two characters of the alphabet). If the number of zeroes and ones is the same $N/2$, then the entropy of the document is $-p_0 \lg p_0 - p_1 \lg p_1$, where $p_0 = p_1 = (N/2)/N = 1/2$, and thus $-1/2(-1) - 1/2(-1) = 1$. The entropy of 1 indicates the number of bits (i.e. 1) needed to encode the two characters.

Example 2 (n equi-probable characters). Consider a document of size N with n equi-probable characters so that $f_i = N/n$ and thus $p_i = f_i/N = 1/n$. Then the entropy of this text is

$$-\sum_{i=1}^{n} p_i \lg(p_i) = -n(1/n)\lg(1/n) = \lg n$$

and thus the n characters can be encoded in $\lg n$ bits. For this reason a character set with 128 characters (e.g. ASCII) uses $\lg 128 = 7$ bits for encoding.

Example 3 (characters with different frequencies). An English text is such a case where the frequency of characters varies. In English text, the probability of a character appearing in the text depends on the appearance of previous characters. For example, given the appearance of character t, it is more likely that the next character in the text will be an h (as in `that, the, this, there, therefore`) than say, a q or a z. Consider now the artificial example of the text being our DNA encoded with characters A,C,G,T, and in the text the corresponding frequency-based probabilities are $p_A = 1/2, p_C = 1/4, p_G = 1/8, p_T = 1/8$ respectively. Then one can use $-\lg(p_.)$ bits to encode these characters, where $_.$ is $_{A,C,G,T}$. This way we could decode A in 1 bit, C in 2, and G, T in three bits each. These generated codes need to form a bit aligned code. In this example, the code generated (and is known as Shannon-Fano code) happens to be the same (in bit length) to the corresponding Huffman code.

Entropy (word-based). One can express entropy and thus the information available by a text in terms of words rather than characters. Consider for example a document that consists not of N characters but we instead use as a metric number of words. Let it have N words. Let the number of distinct words in the document be n. As before let f_i be the frequency of the i-th word and let $p_i = f_i/N$. The (word-based) entropy measure for this document is $E = -\sum_{i=1}^{n} p_i \lg(p_i)$.

Zipf's Law. One important law that applies to words appearing in documents is Zipf's law. Consider words w_i with frequencies f_i (or probabilities $p_i = f_i/N$). Sort these words in descending frequency so that the most frequent word comes first (i.e. it has the highest rank of 1). The word with the lowest frequency will have rank n (since we have n words). Let the frequency of the most frequent word be f (i.e. $f = f_k$ if word w_k is the most frequent one). In the text below instead of dealing with the frequencies of the unordered words we refer to words after they are ranked based on frequencies and thus $F_1 \geq F_2 \geq \ldots$. Thus F_1 is the frequency of the top-ranked word (which used to be word w_k, i.e. $F_1 = f_k$).

Definition 1 (Zipf's Law) *The i-th most frequent word in a text has a frequency F_i such that $F_i = f/i$, where f is the frequency of the most frequent word (and thus $F_1 = f/1 = f$). This also means that*

$$F_i \times i = f$$

Since the text has in total N words of which n are distinct, we conclude that $N = f/1 + f/2 + f/3 + f/4 + \ldots + f/n = f(1 + 1/2 + 1/3 + \ldots + 1/n)$. The series $H(n,1) = H_n = 1/1 + 1/2 + 1/3 + \ldots + 1/n$ is known as the harmonic series of order n. It is known that $H_n \approx \ln n + \gamma$, where γ is Euler's constant and $\gamma \approx 0.5772$. A more general form of this series is $H(n,\theta) = 1/1^{\theta} + 1/2^{\theta} + \ldots + 1/n^{\theta}$. Using the properties of the harmonic series we conclude that $N = f/1 + f/2 + \ldots + f/n = fH_n = f(\ln n + \gamma)$.
In other words, solving for f we get $f = N/(\ln n + \gamma)$, or equivalently for F_i that $F_i = N/(i \cdot (\ln n + \gamma))$. Note that the F_i's form a permutation of the initially unordered frequencies f_i's.

Note. A variation of Zipf's Law is that for example in a company, 10% of the customers generate 90% of the service requests. (Or similarly 10% of the students in a class ask 90% of the questions.)

A reformulation of Zipf's Law is to make it look closer to a Pareto distribution in the denominator. One could consider the previous formulation a simplified version of the standard formulation shown below if one substitutes 1 for θ below.

Definition 2 (Zipf's Law (standard)) *The i-th most frequent word in a text has a frequency F_i such that $F_i = f/i^\theta$, where f is the frequency of the most frequent word (and thus $F_1 = f/1 = f$).*

Then $N = f + f/2^\theta + \ldots = fH(n,\theta)$, and thus $f = N/H(n,\theta)$ and equivalently the i-th ranked word has frequency $F_i = N/(H(n,\theta) \cdot i^\theta)$.

Let $H(n,\theta;q)$ be a variation of the Harmonic series H_n or $H(n,\theta)$ such that $H(n,\theta;q) = 1/(1+q)^\theta + 1/(2+q)^\theta + \ldots + 1/(n+q)^\theta$. Then, we can obtain a Mandelbrot's distribution in which the term i^θ becomes $(i+q)^\theta$ instead.

Definition 3 (Mandelbrot's distribution.) *The i-th ranked word has frequency given by $F_i = \frac{f}{(i+q)^\theta} = \frac{N}{H(n,\theta;q)\cdot(i+q)^\theta}$.*

A negative binomial distribution can be used to express the fraction of the documents containing a word k times.

Definition 4 (Negative binomial distribution.) *The fraction of the documents containing a word k times is given by $F(k) = \binom{\theta+k-1}{k} p^k (1+p)^{-\theta-k}$, where $\theta = 0.42$ and $p = 9.24$.*

Heaps Law on vocabulary size

The size of a vocabulary n can be inferred quite reliably from the size of the document N (we continue using the notation introduced earlier). In order for the result to apply (even empirically), one must assume that **no typing errors occur** that could potentially bloat out the vocabulary size.

Definition 5 (Heaps' Law.) *In an English text with N words, the size n of its vocabulary (i.e. distinct words) is approximately given by $n \approx k \cdot N^{\beta}$. For English texts, $10 \leq k \leq 100$, and $0.4 \leq \beta \leq 0.6$.*

Several times β is considered to be equal to 0.5 and with this value chosen, Heaps' Law takes the form $n \approx \sqrt{N}$.

A set of interesting statistics for the Bible and the TREC collection available at `http://trec.nist.gov/data.html` is the following table. (Note that the terminology in the table below is different than the one we have been using in this section.)

`Text Collection`	Bible	TREC
No of docs : n	31k	741k
No of terms: F	884k	333M
No of disti terms: t	8k	535k
No index pointers: f	701k	134M
Size	4.3MB	2070MB

English texts

We give below a collection of observations that apply to English texts and values for β and θ as those used in Heaps' and Zipf's law. A rule to remember is that $\beta\theta \approx 1$.

```
Average English Word Length (All)              :   4.8 - 5.3 characters     --> w
Average English Word Length (exlude stopwords) :   6.0 - 7.0 characters
Average English Word Length (distinct words  ) :   8.0 - 9.0 characters     --> W
Shortening of text due to stemming             :   5.0 - 6.0 characters, a 30% reduction
```

Text Collection	k	β	$1/\beta$	θ	w	W
AP89	62.95	0.45	2.2			
AP	26.80	0.46	2.1	1.87	6.3	8.0
DOE	10.80	0.52	1.9	1.70	6.4	8.4
WSJ	43.50	0.43	2.3	1.87	5.2	7.4

Other statistics that can be derived from the textbook (Chapter 4, page 78 and onwards) include the following.

The most frequent words of the AP89 collection are `the, of, to, a, and` with $p_i = f_i/N$ ranging from 6.4% to 2.32%. (Note that because these words are sorted by rank already, there is not difference in Table 4.2 of the textbook whether one uses f_i or F_i.) Although ip_i varies from 0.065 to 0.120, for the words with rank 11 through rank 50 the approximation $ip_i \approx 0.093$ applies consistently.

Fact (from Zipf's Law). The product frequency times rank is constant and equal to the frequency of the most frequent work i.e. $f_i \times i = f$. (We use lower-case f for the frequency.) Therefore $f_i i = f$ implies that $f_i = f/i$ and converting frequency into probability of occurrence $p_i = f_i/N$, where N is the total number of words, and thus

$$f_i \times i = f \quad \text{and} \quad p_i = \frac{f_i}{N} \quad \text{implies} \quad p_i = \frac{f_i}{N} = \frac{f}{iN} = \left(\frac{f}{N}\right)\frac{1}{i} = C_N/i,$$

where $C_N = f/N$, and is constant for a given text (of N words).

Question 1. *What is the rank r_m of a word that appears m times ?*

Answer to Question 1. A word that appears m times has frequency m, i.e. $f_t = m$, for some t. Thus the rank of that word would be the index t, the subscript of f, and thus $r_m = t$. Given that, by the previous fact, $f_t \times t = f$ we have $m \times t = f$ and thus

$$\text{Rank of word appearing } m \text{ times: } \quad \text{rank} = t = \frac{f}{m} = \frac{f}{N} \cdot \frac{N}{m} = \frac{C_N N}{m}.$$

Question 2. *What is the highest rank, i.e. the rank of the lowest frequency word? How many distinct words are there?*

Answer to Question 2. The lowest frequency word has frequency equal to 1. Thus $f_i = 1$. Then by way of $f_i \times i = f$ and $f_i = 1$, we have $i = f$. Thus **the highest rank of a word is** f. This also means that there are f **distinct words**.

Question 3 (for information). *What is the lowest rank, i.e. the rank of the highest frequency word?*

Answer to Question 3. The highest frequency word has frequency equal to f. Thus $f_i = f$. Then by way of $f_i \times i = f$ and $f_i = f$, we have $i = 1$. Thus **the lowest rank of a word is** 1**, and it corresponds to the word with frequency** f.

In question 1 we identified the rank of a word that appears m times. There might be more than one words that appear m times. We pick the last one of them to determine the rank. Thus rank r_m is to include all words with ranks $1, \ldots, r_m$ that appear in the corpus m or more times. Also rank r_{m+1} is equivalent to that all words with ranks $1, \ldots, r_{m+1}$ appear $m+1$ or more in the corpus. Thus

Question 4. *How many words appear exactly m times?*

Question 4 (repeated from previous page). *How many words appear exactly m times?*

Answer to Question 4. The number of words appearing m times is the number of words appearing at least m times minus the number of words appearing at least $m+1$ times. The former, by Question 1 is $C_N N/m$ and the latter by Question 1 as well is $C_N N/(m+1)$ respectively. Thus the number of words appearing exactly m times is

$$C_N N/m - C_N N/(m+1) = C_N \frac{N}{m(m+1)} = \frac{f}{m(m+1)}.$$

Question 5. *How many words appearing once in the text?*

Answer to Question 5. The number of words appearing exactly once is by Question 4 for $m = 1$ equal to $f/(1 \cdot (1+1))$ i.e. $f/2$.

Therefore

- The most frequent word has frequency f.
- There are f distinct words in the text; the highest rank value in the text is f (the lowest rank value of 1 corrsponds to the word with the highest rank and frequency).
- There are $f/2$ words of frequency 1.

The first step of document processing involves a number of text operations or transformations. They can be grouped into three phases: (a) the lexical analysis or parsing or tokenization, (b) the linguistic analysis, and (c) the indexing. (The first phase and also the second phase has already been discussed in some detail in Subject 1/2.)

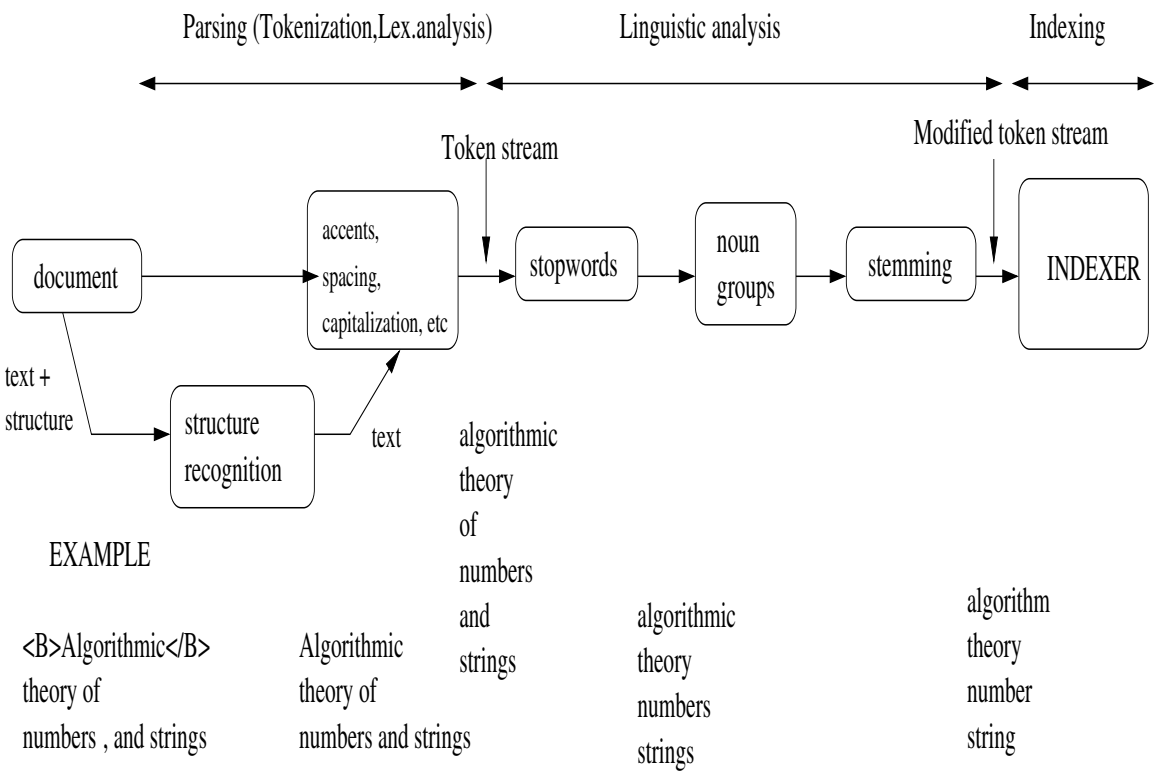

Figure 1: Document Processing

A. Parsing. The first phase is known by a variety of names such as **lexical analysis** and **parsing** and involves the **tokenization** of the input document. For this reason it is sometimes also called tokenization. This phase thus converts the character stream of the input document(s) into **tokens** (the use of the common **words** might exclude number, dates, etc from inclusion into tokens). The objective is to determine which ones of the tokens can serve as keywords and finally generate **index-terms** out of them. It is the **index-terms** that will be stored in the index, not (necessarily) words or tokens. (The notion of a keyword is more theoretic: it identifies the subset of the words/tokens that are considered important in a document.) Parsing also includes a number of processing steps that need to resolve several issues such as treatment of number, dates, case (upper vs lower), accents and punctuation, fonts.

This phase consists of a sequence of **subphases** that can include the following: (a) File-type that determines syntax, presentation style, semantics, including character encoding (ASCII vs Unicode), and document language (English vs Germanin languages vs etc), (b) white space removal is decided and phrases are handled (eg. is United States treated as two words separately or one), and short words are resolved (XP, NT, WWII), (c) case folding is determined and resolved and accents and punctuation marks are dealt with, (d) hyphenization is resolved, (e) Positional Information of word (aka context) that also ignores tag words to generate offset (i.e. `<TITLE>` does not increment offset counter), (f) Title Info is recorded (surrounded by `TITLE` tags), (g) Font-Type Info and Size is noted (e.g. text is `between <B> and </B>`) or what the font-size is relative to the default one of the page, (h) Header text, i.e a distinction between (`H1,H2,H3`) and possibly (`H4,H5,H6`) is being made and recorded, (i) List Info, such as `UL, OL, DL, LI` is utilized, (j) and URL is properly attributed and extracted from Anchor along with associated text. than the SourceURL, and (k) number and date translation is decided.

As a result of this first phase, the tokenizer might return not just the token identified (say `algorithm`) but also metadata information about it other than its value (name). For example if it is in an HTML document, for `algorithm`, its appearance in the HTML document becomes important and thus it will be transmitted by the tokenizer (i.e. that say it is in bold-face font or not, etc). These first text transformations have a simple linguistic and context-based element. They identify issues such as what will become a token, and whether position within a document is important or not.

B. Linguistic Analysis This phase is applied to the output of the previous phase (the token stream) or is combined with it. In this phase language-related text transformations are applied to generate the index terms. The set of text transformations involve the application of a list of `stopwords`, `stemming`, that will further restrict or modify the stream of tokens that would generate possible keywords and eventually index-terms. Sometimes an index-term is a token (word) that does not appear per se in the text or it appears there in a completely different form (with a suffix). For example with reference to Figure 1, `algorithm` is an index-term, although the word `algorithm` never appears in the text: the input document contains only the word `algorithmic`.

- **1. Stopwords.** Frequent not very useful words are poor index-terms and bloat indexes. These frequent words include articles and connectives. However important text such as `to be or not to be` might contain such frequent words and thus one want to be able to index it properly. Will this text be missed because of stopword restrictions or improper choice of the stopword list? Such a case needs to be avoided.
- **2. Stemming**. A stem is the substring of the word left after the removal of its affixes (prefix and suffix). The stemming operation attempts to reduce distinct words to their common grammatical root.
  - **2a.** Stemming eliminates duplicates of the same word (e.g. in `algorithm`, `algorithms`, `algorithmic` or the more involved `swimming`, `swam`, `swim`) whether they are in plural or singular form, in gerund form, or have past tense suffixes.
  - **2b.** To stem or not to stem? (e.g. Harman(1991), Frakes (1992)).
  - **2c.** Algorithms for stemming include Porter's and Harman's with the latter being described on the following page.
- **3. Lemmatization.** It refers to the grouping of different inflected forms of a word (e.g. `car, cars, car's, cars'`, and also `good, better, best`).
- **4. Synonyms.** Similar in meaning words. (`car`, `vehicle`, `automobile`) or the not so obvious (`metro` vs `underground` vs `subway`).

- **5. Noun groups**, (e.g. `kitchen table`). Sometimes noun groups can be identified during the previous phase when dealing with specific phrases.
- **6. Spelling.** `center` vs `centre` . Or the more challenging `towards` vs `toward`. Is one of them a typo? Do we apply lemmatization?
- **7. Typing Errors.** Do you correct them ? How does one handle them? Usually at query time correct spelling by checking each word against the lexicon for distance editing (definition on page 16) of at most 2-3.
- **8. Thesaurus use.** Consider the case of a `thesaurus` for different subjects that determine the list of index terms as a subset of the tokens identified so far. Most search engines assume that the set of all tokens of all documents (after stemming and stopword elimination) is the thesaurus. Identify synonyms, words with similar meaning and search for all of them if one in the query list.

Stopword list. A list of possible stopwords can include a subset of the following words. One reason we use stopwords and decide not to index them is storage/space problems. The inverted index might grow too large beyond our storage capabilities if we index several very frequent words. If space is not an issue one can decide not to treat some or all of these words as stopwords (and thus index them).

```
 I a about an are as at be by com en for from
how in is it  of on or that the this to was what
when where who will with www
```

It is tempting to consider the ordinarily disjunctive word `or` a stopword. But be reminded (again) that `OR` might be the designator for the state of Oregon, or an acronym for Operations Research or for Operating Room. It is imperative that the stopword list be kept as short as possible. Also phrases such as `it is, there is, up to us` include several potential stopwords; however a `us` might be a writing for `USA` as well.

At this level, a semantics-based classification of a Web-page might also take place that characterizes it for example as `sports`, `politics`, `academic`, etc.

There are some arguments in favor or against stemming e.g. Harman(1991) and Frakes (1992). If there is plenty of space available in the search engine architecture, stemming might be eliminated for certain words (and thus `algorithm`, `algorithms` and `algorithmic` might give different sets of results, if searched for).

There are several algorithms for stemming that are simple and also other algorithms that are rather more complicated. For example Porter's algorithm and Harman's are two such methods. The latter is quite straightforward as shown below. A third widely used algorithm is an one by Krovetz.

Harman's stemming Algorithm.

```
 HarmanStemmingAlgorithm(word)
1. If word ends in -ies but not -eies or -aies
    then -ies --> -y;
2. If word ends in -es  but not -aes, -ees or -oes
    then -es  --> -e;
3. If word ends in -s but not -us or -ss
    then -s   --> -;
```

The algorithm is not perfect, yet it is quite simple. However it has its limitations. What happens for example if the input is `tries, retrieves, foxes, dies` ?

In general it has been observed by Porter through the experimental use of his own stemmer, that the size of a vocabulary can be shortened by 30% if one uses stemming. (One phase, but not the complete algorithm of Porter's stemming method is available in the textbook on page 92.)

Note. The parser should also be capable of accommodating errors including syntax errors and thus be able to deal with free-form text in which no specific syntax is implied. Thus if there is a syntax and semantics implied in the text the parser should be able to take advantage of this additional information. In the absence of such information it should also be able to work properly and miss as few words as possible, since a word missed might be an index-term missed after all.

Phase B: Linguistic Analysis and word similarity

Two documents or two words can be compared to determine their similarity.

- One way to measure similarity is by using a **distance function**. The Hamming distance of say two strings of the same length is the number of positions in which they are different. Distances satisfy the triangular inequality $d(a, c) \leq d(a, b) + d(b, c)$.
- The **edit** or **Levenshtein** distance is the minimum number of character insertions, deletions, and substitutions we need to perform in any of the strings to make them equal. The edit distance of `center` and `centre` is two and of `color` and `colour` is one.
- Another measure is the longest common sequence (deletion of characters is only allowed). For example `color` and `colour` have `color` whereas `survey` and `subway` have `suy`.

n-gram. An n-gram is a sequence of n words that have a special meaning if they occur together.

According to the textbook (page 101) the following data related to n-grams can be obtained from Google's text collection.

```
          Google Text  Collection
Number of words   n   :  1   trillion
Number of 1-grams     : 13   million
Number of 2-grams     : 314  million
Number of 3-grams     : 977  million
Number of 4-grams     : 1.3  billion
Number of 5-grams     : 1.17 billion
```

Figure 2: Google-reported n-gram information

Note. The following might also be deemed quite surprising as far as n-grams or phrases are concerned.

Most frequent ENGLISH phrase (or 3-gram) is: `All rights reserved`.

Most frequent CHINESE phrase (or 3-gram) is: `Limited Liability Corporation` (LLC).

3. Indexing The token stream generated after the end of the second phase for a given input document, will be merged with those of other documents to generate a forward index. In a `forward index` for every document the words that appear in the document are listed along with other word attributes. Eventually, when the forward index is generated completely, an `inversion` operation will be performed to eventually generate an `inverted index`. This inversion operation is usually a sorting operation that is stable. A sorting algorithm is stable if the relative order of same-valued keys is maintained between the input and output. This inversion is shown in Figure 3. The original token streams of three documents are merged into a single token stream. In this token stream, an ordering exists based on docID (primary key) and also within a document based on the word offset attribute that is assumed to be the only attribute for this example (secondary key). When the inversion operation is applied, the stream is sorted in a stable manner based on wordID (primary key), then within the same wordID based on docID (secondary key), and if the word appears more than once in a given document, based on word offset (secondary key). The ordering based on the two secondary keys (docID, word offset) is already available in the input. Thus we would like it not be destroyed during the inversion. A choice of a stable sorting algorithm will achieve this; if one can not use a stable sorting algorithm, an algorithm can be made to behave as if it was stable by adding to each key additional information that will make it unique (and thus eliminate the precondition of having same valued keys to determine stability).

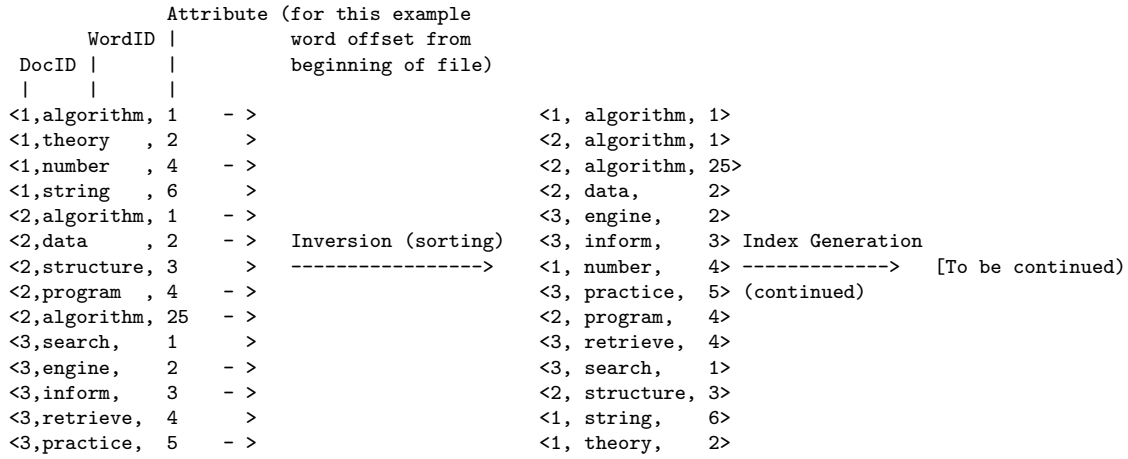

Figure 3: Inversion of a forward index to generate an inverted index