Chapter 2:
Terms, Vocabulary, Postings

Reading:

- 2.1
- 2.2, stop at 2.2.2

Recall the major steps in inverted index construction:

1. Collect the documents to be indexed.
2. Tokenize the text.
3. Do linguistic processing (normalization)
4. For each token, add the document ID to the term's postings list (Index the document)

Characters and documents

Character sequence:

Files are a sequence of **bytes** which must be converted into a sequence of **characters**.

- Characters are **encoded** using a **code space**. Each character is given a **code point** in that space. Example: "Hello" in ASCII is 0x48656C6C6F.
  - Should we assume that all files are encoded in ASCII? Probably not. Can we detect this at run-time?
  - What encodings do your favorite programming languages use?
- Characters may need to be decoded out of some binary representation – compression (ZIP), DOC.
- Other decodings: in HTML or XML, decoding entities with `&...;`
- Text is not always LINEAR: Arabic text takes on two-dimensional and mixed order characteristics.

Example 1. Arabic text from page 21. RTL but interspersed LTR.

Document unit:

We assumed that a **document** is a **file**, but that does not have to be the case.

- An email program often stores many emails in one file.
- Some documents (web pages) are split into multiple files.

For long documents, **granularity** becomes an issue:

- Search a collection of books for "angels" – maybe "baseball" is on page 1, and "angels" is on page 1000.
- Could index smaller units as individual documents: paragraphs, sentences. Tradeoffs?

Example 2. What effect does granularity have on precision and recall?

These are issues that must be considered as part of any search engine solution.

Determining the vocabulary

Tokenization: taking a sequence of characters and breaking it into meaningful pieces called **tokens**, maybe removing punctuation and whitespace.

- So a **token** is a sequence of characters from a file
- A **type** is a distinct string from the corpus
- A **term** is a representation of a type that is included in an information retrieval system. (After processing. Lowercasing, stemming, lemmatizing, ...)

Example 3. Give the tokens, types, and terms for the document `Marley joined me to watch "Marley and Me."` Use lowercasing and suffix elimination for terms.

How to tokenize?:

- Naive: split on whitespace, throw away punctuation.
- Apostrophes for posessives and contractions: `angel's`, `angels'`, `O'Neill`, `aren't`
- Unusual tokens: **ip addresses**, web URLs, package tracking numbers
- **Hyphenation**: splitting up vowels in words (co-education), joining nouns as names (Hewlett-Packard), word grouping ("he's a stand-up guy")
- Even whitespace maybe shouldn't split tokens. `Los Angeles` – one or two terms? Borrowed foreign words – `ad hoc`? Optional groupings: `white space` vs `whitespace`. Phone numbers. Dates. Airlines: `San Francisco-Los Angeles`
  - Search for York University, get New York University
- Other languages: use of `l'` for "the" in French. Compound nouns in German: **Lebensversicherungsgesellschaftsangestellter - life insurance company employee.** East Asian language: no spaces!!
- **THESE THINGS MATTER FOR INDEXING AND QUERIES**
- Golden rule: whatever you apply to documents, also apply to queries

Stop lists:

Some words are too common to be useful in a search: **stop words**. So remove them... what could go wrong?

- `President of the United States` is much more precise than `President AND United States`
- `To be or not to be`
- Trend: few if any stop words.

Normalization:

Canonicalize tokens so that matches occur despite superficial differences in character sequences. Example: match `USA` and `U.S.A.`

- Implicit equivalence classing: automatically search for related spellings (removing hyphens, apostrophes)
- Hand-constructed token relations: synonym lists: `car -> automobile, auto`. Can be asymmetric. How to incorporate into a IR system?
  - `windows -> window, Windows`
  - `window -> windows`

Accents: remove? In English, not much meaning. In Spanish: `pena` (sorrow) vs `penya` (cliff)

Stemming

Stemming: removing common suffixes from English words, reducing them to their *stem.* Stemming can reduce the size of the vocabulary, and cause normalization among difference suffixes of the same stem, but sometimes loses full meanings of words.

Porter stemmer:

A consonant in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. For example, the word "boy" has two consonants, 'b' and 'y', while "bye" has one consonant 'b'. We denote consonants by the symbol c, and variables by the symbol v. Any sequence of one or more c symbols will be denoted by C, while any sequence of one or more v symbols will be denoted by V.

Example 4. Give the capitalized symbol sequences of the words tree, attempt, synergetic, alabama, and sushi.

In general, every word is of the form $C^*\,(VC)^m\,V^*$, where a * represents "may or may not be present" and m represents "m or more" for $m \geq 0$. m is called the *measure* of a sequence.

Example 5. Give the measures of the words from Example 5.

The Porter stemming algorithm uses rules to decide on when a suffix can be removed, and what, if anything, it will be replaced by. The rules have the form

$$(\text{condition})\; S_1 \rightarrow S_2$$

where, if condition is satisfied by the **stem** that precedes S_1 , then S_1 is replaced by S_2.

Conditions are usually in terms of m (e.g., $m > 0$), but also can be:

- $*S$: the stem ends in S (similarly for other letters)
- $*v*$: the stem contains a vowel
- $*d$: the stem ends with a double consonant
- $*o$: the stem ends with (lowercase) *cvc,* where the second c is NOT W, X, or Y

The algorithm proceeds in 5 steps. In each step, only one rule can be matched, and the rules are organized so that the FIRST rule that matches is the one used.

Example 6. Apply the Porter stemming algorithm to the words caresses, plastered, hopping, filing, happy, condition, conditionality, rational, rationality, decisiveness, electrical, electrocution, adjustment, adjustable, probate, probationary, and controlling.

Phrase Queries

Reconsider searching for "New York University". If treated as AND query: New AND York AND University – what could go wrong?

Suppose we have this corpus:

- New York University Ranked Best in State
- York University Opens New Science Lab
- Increasing Debt Among University Students in New York

A **phrase query** is a query for documents containing a sequence of tems **in order** in the document text. To support phrase queries, it is no longer enough to record presence of a term in each document; we must somehow capture positional information.

Biword index:

A **biword index** is a separate inverted index that records **biword phrases**. For each pair of adjacent terms, add an index entry for the pair and the document ID. To satisfy a phrase query with a biword index, build biwords from the query and use the index.

Example 7. Construct a biword index for the documents above. Answer the query "New York" and "New York University" with the biword index.

Example 8. Brainstorm issues with the biword index for general phrase queries.

- Tri-words / k-words?
- Give an example of a document that will give a false positive for "New York University"
- Effect on vocabulary size

So a biword index is nice, but some issues.

Positional index:

Biword index not the standard solution. A **positional index** stores postings as a document ID with a list of *term positions* where that term occurs in the document. We also store the term frequency for use later.

To answer a phrase query, start the normal intersection merge. When two lists have the same document ID, do another intersection on the positions, looking for positions that are +1 from each other.

Example 9. Build a positional inverted index for the documents. Answer "New York University".