**Indices and Boolean retrieval**

**Precision**

* Fraction of the retrieved docs that are relevant to the user’s information need.

**Recall**

* Fraction of the docs in collection that are retrieved.

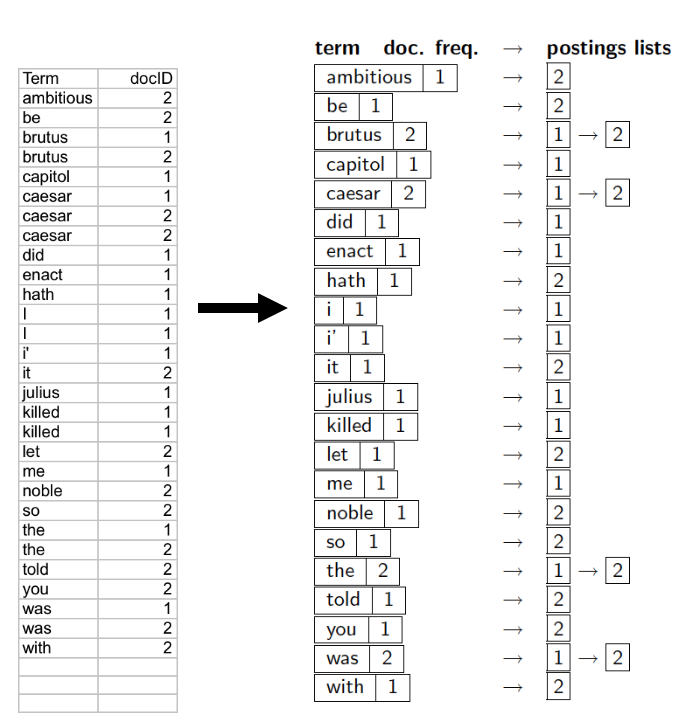
**Term-document incidence matrices:**

* Can generate incidence vectors for each term.
* Answer queries by bitwise AND/OR operation.
* Impractica l in reallife due to sparseness.

**Inverted index**

* For each term t, we must store a list of all documents that contain t. (Posting list)
  + This list is sorted by docID.
* Identify each document by docID, a docuement serial number.

**Inverted index construction**

* **Initial statges**
  + Tokenization
    - Cut character sequence in to word tokens.
  + Normalization
    - Map text and query term to the same form. (**U.S.A** and **USA**)
  + Stemming
    - We may wish different forms of a word root to match. (**Authorize, authorization**)
  + Stop words
    - We may omit very common words. (**the, or, a, of**)
* **Token sequence processing**
  + Sort by terms and then by docID.
* **Construct dictionary and postings**
  + Multiple term entries in a single document are merged.
  + Split into dictionary and postings.
  + Document frequency information is added. (Number of different documents that contains a term)

**Query processing**

* Retrieve the posting lists of the query terms.
* Merge the posting lists together. (AND/OR)
  + AND: Construct a new posting list that only contains the postings that are in both.
  + OR: Construct a new posting list that contains all of the postings.

**Boolean retrieval model**

* The **Boolean retrieval model** is being able to ask a query that is a Boolean expression
  + Boolean Queries are queries using AND, OR and NOT to join query terms.
    - Views each document as a set of words.
    - Is precise: document matches condition or not.
  + Perhaps the simplest model to build an information retrieval system on.
* Primary commerial retrieval tool for 3 decades.
* Many search systems are still using the Boolean retrieval model:
  + Emal, library catalog, etc.

**Query evaluation optimizations**

* Process query terms in order of increasing frequency/length
  + Eliminate most of the postings during the ealier stages of the process.
* General optimization
  + (A OR B) AND (C OR D)
  + Get document frequency for all terms
  + Estimate the size of each OR claus by the sum of its document frequencies.
  + Process in increasing order of OR clause sizes.

**Phrase queries**

* We want to be able to answer queries such as ‘’Standford university’’ as a phrase.
* Thus the sentence ‘’I went to university at Standford’’ is not a match.
  + The concept fo phrase queries has proven easily understood by users; one of the few ‘’advanced search’’ ideas that works.
  + Many other queries are implicit phrase queries.
* For this. It is no longer siffices to store only <term: docs> entries.

**Solution 1: Biword indexes**

* + Index every consecutive pair of terms in the text as a phrase.
  + For excample the text ‘’Friends, Romans, Countrymen’’ would generate the biwords:
    - Friends romans
    - Romans, countrymen
  + Each of these biwords is not a dictionary term
  + Two-word phrase query processing ios now immediate
  + **Longer phrase queries:**
    - Longer phrases can be processed by breaking them down
    - ‘’Standford university palo alto’’ can be broken into the boolean query on biwords
      * **Standford university** AND **university palo** AND **palo alto**

**Issues with biword indexes**

* + False positives
    - Without the document contents, we cannot verify that the documents matching the boolean query do contain the phrase. The document could contain all the biwords , but not as a coherent phrase.
  + Index blowup due to bigger dictionary
    - Infeasible for more than biwords, big even just for biwords
  + Biwords indexes are not the standard solution, but can be part of a compound query evaluation strategy.

**Solution 2: Positional indexes**

* + In the postings, store for each term, the positions in which the tokens of i appear.
    - Term : frequency,
      * Doc1: pos1, pos2...
      * Doc2: pos1, pos2...
  + A position is the order of the token, not the index in the content of the document. Postition 15 means the token is the 15th token in the normalized document content.
  + **Processing a phrase query**
    - Phrase **‘’to be or not to be’’**
    - Terms: to, be, or, not
    - Extract inverted index entries for each distinct term
    - Merge their doc:position lists to enumerate all positions with ‘**’to be or not to be**’’.
      * To: 2: 1,17....., 4: 8, **16**, 190.......
      * Be: 1: 17...., 4: **17**, 197......
    - Same general method for proximity searches.
  + **Proximity queries**
    - **LIMIT /3 STATUTE /3 FEDEAL /2 TORT**
      * /k means ‘’within k words of’’.
    - Clealy, positional indexes can be used for such queries, biword indexes cannot.
  + **Positional index size**
    - A positional index expands postings storage substantially
      * Even though indices can be compressed
    - Nevertheless, a positional index is not standardly used because of the power and usefulness of phrase and proximity queries, whether used explicitly or implicitly in a ranking retrieval system.
    - Need an entry for each occurrence, not just once per document.
    - Index size depends on average document size
      * Average web page has < 1000 terms
      * SEC filings, book, even some epic poems, easily 100,000 terms.
    - **Rule of thumb**
      * Positional index is 2-4 time as large as a non-positional index
      * Positional index size is 35%-50% of volume of original text.
      * **(For all ‘’English like’’ languages)**

**Combination schemes**

* + Biword indexes and positional indexes can be profitably combnined
    - For particular phrases, such as named entities (‘’Michael Jackson’’, ‘’Britney Spears’’) it is inefficient to keep on merging positional posting lists.
      * Even more so for phrases like ‘’The Who’’(band)