1.1 An example information retrieval problem

Let say, we want to built a document collection of novel books. To do that, we need to create a matrix of terms and documents. If we found one billion terms in one thousand documents, our document collection we be represented as matrix of one billion rows multiply by one thousand coloums. If an document d_y included a term t_x, then we added value **1**, else we added **0**. So, our document collection will be an binary matrix.

In the we need to find any document that matched with query, for example “Harry Potter The Gandalf ”. We may used boolean logic that translate our query into “Harry AND Potter AND The AND Gandalf”. Technically, we send a command to find any documents that exactly included terms “Harry”, “Potter”, “The”, and “Gandalf”. If a document does not has a term “Harry”, but has rest of three terms, then a documents is not match. For example: we have five documents and the boolean operation will be:

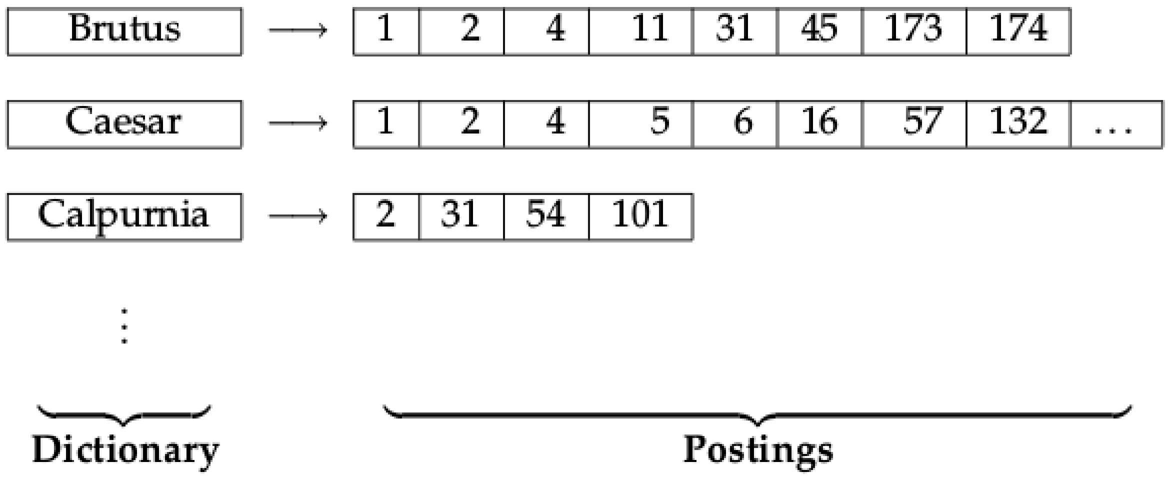
$$10010 \text{ AND } 01010 \text{ AND } 00010 \text{ AND } 10111 = 00010$$

We can said that only document with id four (d_4) is matched to our query.

1.2 A first take at building an inverted index

If we have very big document collection, then using matrix to store our terms and documents id is efficient, since we store value 0 for every not matched terms. A good solution is by only store matched terms. One good data structure that make it possible is inverted index. In the real world, you can find similar implementation in the last page of book called as “Index” where some important terms ordered alphabetically and for each term included page number where we can found its term. In the computer science, we use linked list data structure to implement an inverted index.

The building of an inverted index will looked like picture below:


Brutus → 1 2 4 11 31 45 173 174
Caesar → 1 2 4 5 6 16 57 132 …
Calpurnia → 2 31 54 101
Dictionary
Postings

Dictionary is set of terms found in collection sorted alphabetically and **Postings** is set of sorted document identifier that found in specific term.

1.3 Processing Boolean Queries

Processing an boolean queries required an merge algorithm to find union for disjunction operation and intersection for conjunction operation. We could optimize query processing through order boolean execution by postings size of term.

1.4 The extended Boolean model versus ranked retrieval

Boolean retrieval model have been widely used in search engine system before 1990. Westlaw is largest commercial legal search service since 1975 which adopted extended boolean model as main search engine until 1992. Below the examples of boolean model querying:

- "John AND Drive is same sentence": John /s Drive
- "John AND Drive is same paragraph": John /p Drive
- "John OR Joe": (John Joe)
- "Any term consisted post-": post!

However, extended boolean model still not capable to rank retrieved information because the essential of boolean model is only capable to identify any terms in query found in any documents. Boolean query can not judge how similar each document and how relevant each document to information need.