**Final Project**

To complete the Project I have adopted the following approach:

Firstly, implement an inverted index with a data structure consisting of the following:

* An array storing the occurrence lists of the terms (in no particular order)
* A compressed trie for the set of index terms, where each external node stores the index of the occurrence list of the associated term.

Given a query, we use the index to return the list of documents relevant for this query. The inverted index contains mappings from terms (words) to the documents that those terms appear in. Each vocabulary term is a key in the index whose value is its postings list. A term’s postings list is the list of documents that the term appears in.

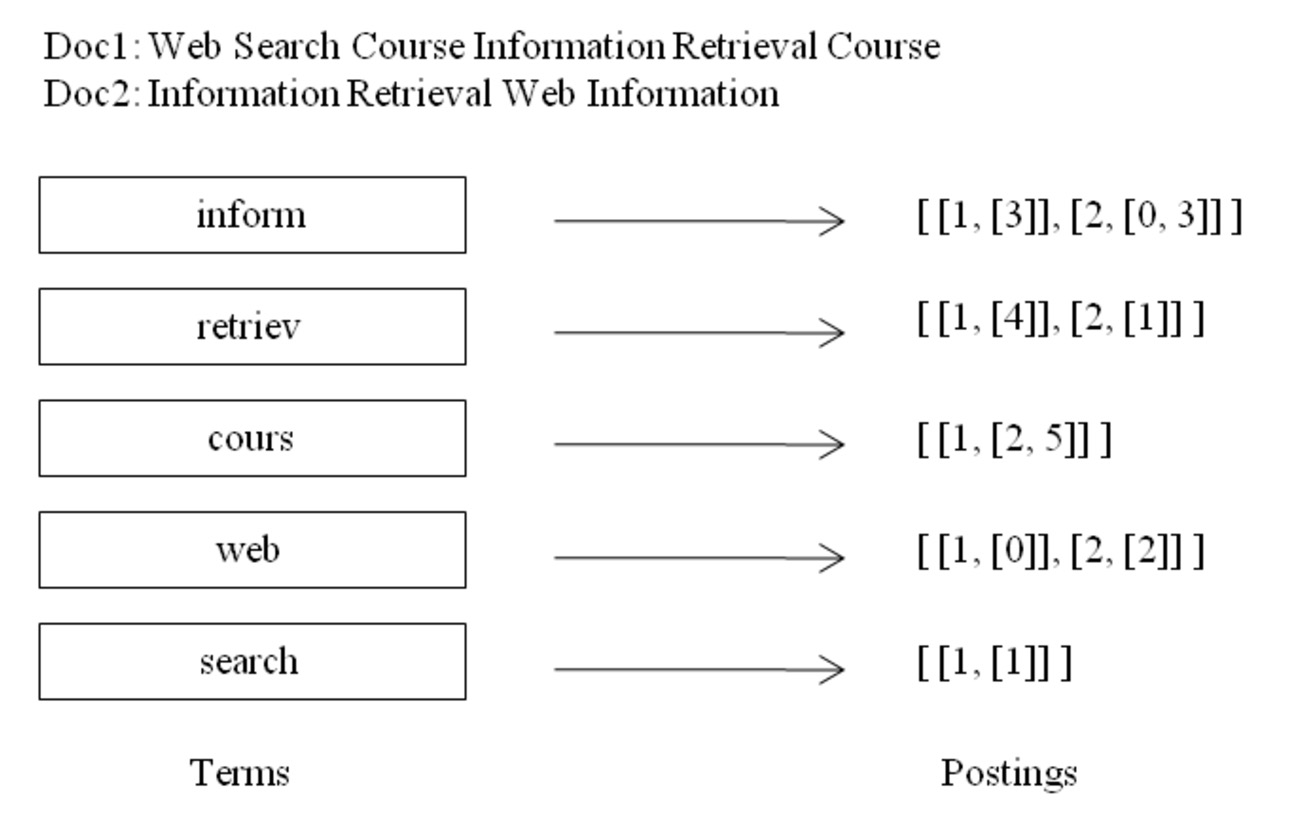
**Query Types**

The current search engine handles only one work queries

One Word Queries: Queries that consist of a single word. Such as movie, or hotel.

**Parsing the Collection**  
While parsing the document collection we will decide which words will be the terms in the index. As mentioned above, we don’t want every word to be a term. So, while parsing the Wikipedia articles we will perform the following operations on each page in this order:  
1) Concatenate the title and the text of the page.  
2) Lowercase all words.  
3) Get all tokens, where a token is a string of alphanumeric characters terminated by a non-alphanumeric character.

**Building the Inverted Index**  
The inverted index is the main data structure of our search engine. We will use a TreeMap to store the inverted index in memory. The reason is we will perform lots of lookups (one for every term in the document), and we will also add lots of keys (every term is a key), so we want these operations to be very efficient. Since Maps have average O(1) lookup time and amortized O(1) insertion time, they are very suitable for our needs.



**Trie Based Implementation**:

When multiple keywords are given and the desired output is the pages contain- ing *all* the given keywords, we retrieve the occurrence list of each keyword using the trie and return their intersection. To facilitate the intersection computation, each occurrence list should be implemented with a sequence sorted by address or with a dictionary, which allows for a simple intersection algorithm similar to sorted sequence merging

