Inverted Index

October 19, 2012

1 Introduction

Today, top search engines like Google and Yahoo use a data structure called **Inverted Index** for their matching of queries to the documents and give users the relevant documents according to their rank. Inverted Index is basically a mapping from a word to its position of occurence in the document. Since a word may appear more than once in the document, storing all the positions and the frequency of a word in the document gives an idea of relevance of this document for a particular word. If such an inverted index is build up for each document in the collection, then when a query is fired, a search can be done for the query in these indexes and ranking is obtained according to the frequency.
Mathematically, an inverted index for a document D and strings $s^1, s^2, ..., s^n$ is of the form

$$\begin{gathered} s^1 -> a_1^1, a_2^1, ... \\ s^2 -> a_1^2, a_2^2, ... \\ . \\ . \\ . \\ s^n -> a_1^n, a_2^n, ... \end{gathered}$$

where a_l^k denotes the l^{th} position of k^{th} word in the document D.

To build up this kind of data structure efficiently, **Tries** are used. Tries are a good data structure for strings as searching becomes very simple here with every leaf node describing one word. To build up an inverted index given a set of documents using trie, following steps are followed -

- Traverse one document and insert words into a trie. As a leaf node is reached, assign it a number (in increasing order) representing its location in the index (staring from 0). Add the position of this word into the index.
- Now for a word which occur more than once in the document, when attempt for second insertion into the trie is made, a leaf node already containing that word would be found and its value would tell the location in the index. So simply go to this index and add another position for this word.
- Do this till end of document is reached. Now, you have a trie and an inverted index for the first document.
- Repeat this procedure for the rest of the documents.

Now follow the below steps to search for a word from the inverted indexes and tries of all the documents -

- For every document, first search for the word in the corresponding trie and get its location in the inverted index of that document.
- Then traverse through all the positions and see which document has most frequency and arrange the documents accordingly (in decreasing order).

Also, in every document there are special words called "anchor texts" which have more importance than a normal text word. For example – a download link. So for the same word, its occurence as an anchor text increases the relevance of that document over its normal occurence.

2 Problem Statement

For this assignment, you need to create an inverted index for a collection C of documents from 1 to n. Every document will be a plain text file with first line storing its id from 1 to n and next few lines containing space or new line separated words. The index should be an array of lists with size of array equal to total number of distinct words in the array and the list for each word contains the locations of the word in the document. The trie used for this construction can be represented in any form (array/linked list/trees etc.). So you would have n such tries and inverted indexes. Then you should ask user for the queries (single-word) and give the order of documents in decreasing order of relevance. For our case, the anchor texts are represented by following the word with a $*$. So if you have something like - *"Rats fear cats and cats* fear dogs."* then here 1^{st} *cat* is a normal word whereas 2^{nd} *cat* is an anchor text. So now your array size will be $2 * total number of distinct words$ in the document as you would store positions of normal text and anchor text separately for a given word. And now relevance should first be decided by the frequency of anchor texts and within them collision should be resolved by frequency of normal text.

```
D1      1
        it is what it is
D2      2
        what is it
D3      3
        it is a banana
```

Below are the corresponding tries and inverted indexes for the 3 documents (figure 1).

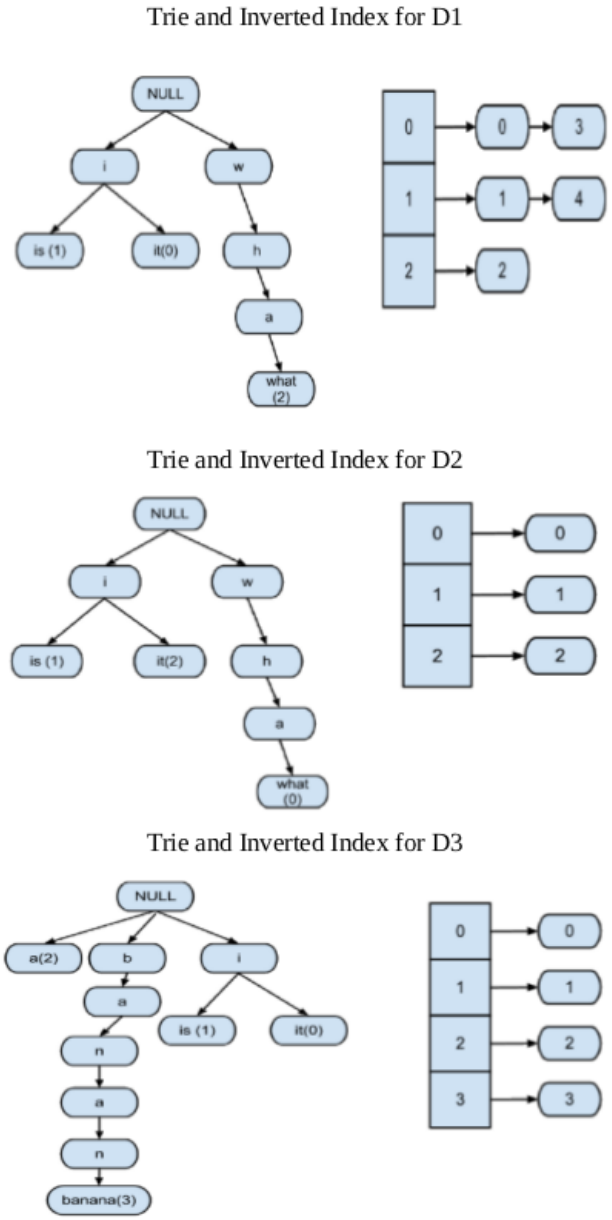

Figure 1: Trie and Inverted Index for Documents 1, 2 and 3

Now if query is "it" - then search in 1^{st} index gives – $0, 3(freq = 2)$, 2^{nd} index gives $2(freq = 1)$ and 3^{rd} one gives $0(freq = 1)$. So, our output is – $1, 2, 3 or 1, 3, 2$ (as document 2 and 3 have equal relevance).

NOTE

- The names of the data files should be taken from command line. After

building the inverted index, you should ask for query again from command prompt and also give an option of quitting any time the user want.

- The inverted indexes should be written to files named as "1...n.txt" with each line corresponding to one word in the document.
- You can ignore case-sensitive words i.e., Cat and cat are same.
- Also ignore symbols in the text (if any) like .,-?