Deepak Kandepet

MENU

Dissecting Lucene – The Index format

posted in **internals** on **October 1, 2011** by **deepak** SHARE

"Who is putting the math books in the horror section?"

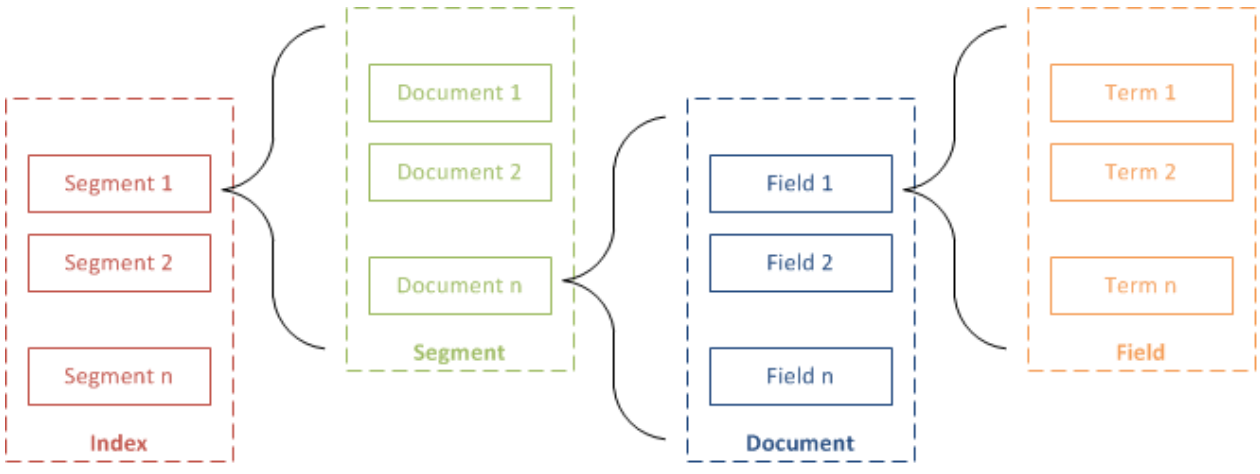
Lucene is high-performance, scalable, full-featured, open-source text search engine written in Java. Since I am a search engineer by profession, I wanted to learn more about Lucene and its internals.

This article is about the index format of the 3.4 Lucene. Specifically the Lucene inverted index.

Lucene Inverted Index

Some Definitions

- **Index:** An Index is basically a set of documents that are to be searched. The index may be composed of multiple sub-indexes, or segments. Each segment is a fully independent index, which could be searched separately.
- **Fields:** When a document is added to an index, different sections of the document are identified and are given common names called fields. e.g. Document title, document body, hyperlinks, the document URL, etc.
- **Terms:** The words in the fields are extracted and are stored in the index as terms. A term is basically a string extracted from the document.
    - The same string in two different fields is considered a different term.
    - Terms are represented as a pair of strings, the first naming the field, and the second naming text within the field.
    - **Stored/Indexed**:
        - In Lucene, fields may be *stored*, in which case their text is stored in the index literally, in a non-inverted manner.
        - Fields that are inverted are called *indexed*.
        - A field may be both stored and indexed.
    - **Tokenized/Non-Tokenized** :
        - The text of a field may be *tokenized* into terms to be indexed, or the text of a field may be used literally as a term to be indexed.
        - Most fields are tokenized, but sometimes it is useful for certain identifier fields to be indexed literally.



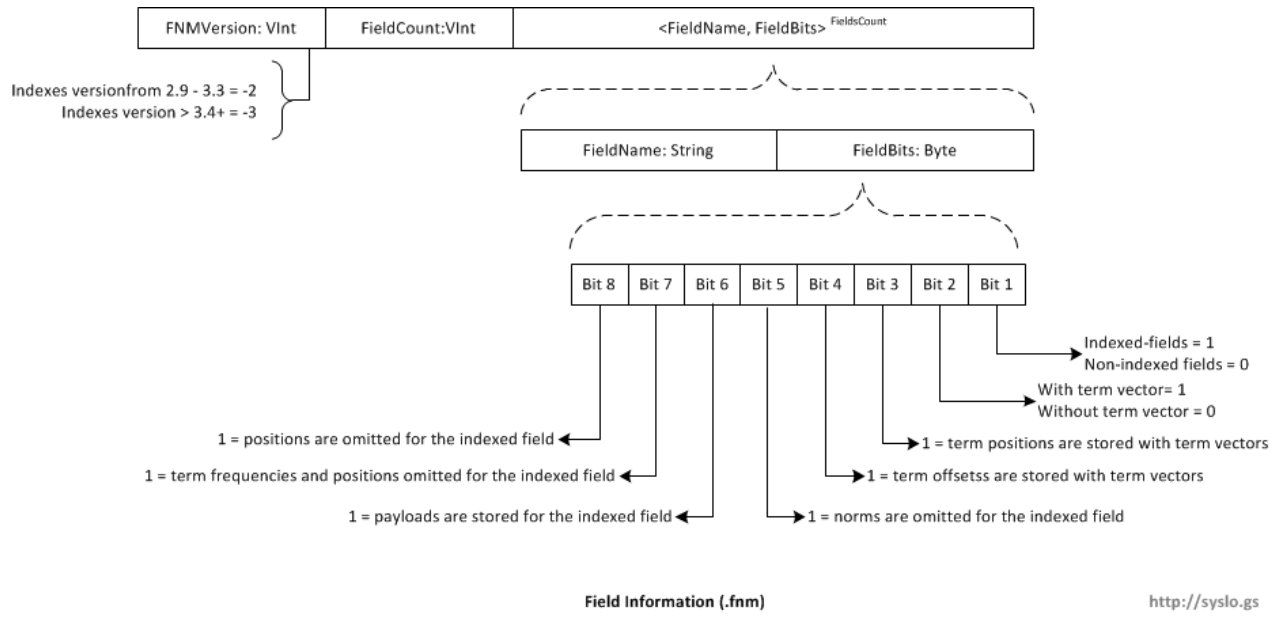
Index Structure http://syslo.gs

Lucene uses file extensions to identify different parts of the index. Here is an example of an index generated on my machine:

```
-rw-r--r-- 1 root root  1122 Oct  4 20:23 _0.fdt
-rw-r--r-- 1 root root   132 Oct  4 20:23 _0.fdx
-rw-r--r-- 1 root root    32 Oct  4 20:23 _0.fnm
-rw-r--r-- 1 root root 10592 Oct  4 20:23 _0.frq
-rw-r--r-- 1 root root    20 Oct  4 20:23 _0.nrm
-rw-r--r-- 1 root root 32168 Oct  4 20:23 _0.prx
-rw-r--r-- 1 root root   313 Oct  4 20:23 _0.tii
-rw-r--r-- 1 root root 18587 Oct  4 20:23 _0.tis
-rw-r--r-- 1 root root   271 Oct  4 20:23 segments_1
-rw-r--r-- 1 root root    20 Oct  4 20:23 segments.gen
```

Field Information (.fnm)

Field info file (with suffix .fnm) records the index time attributes and field names for every field. Fields are numbered by their order in this file. Thus field zero is the first field in the file, field one the next, and so on. Note that, like document numbers, field numbers are segment relative.



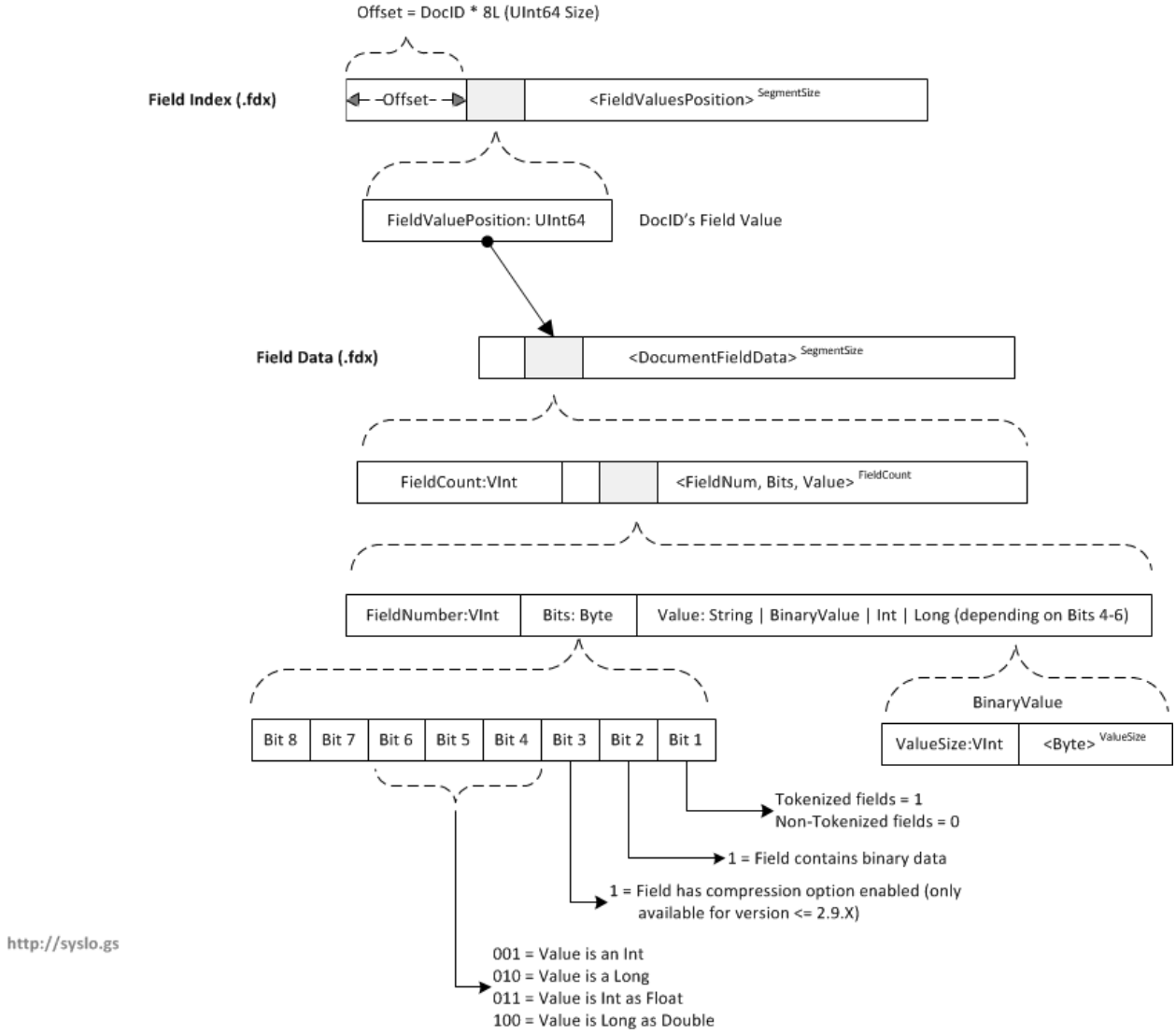
Field Information (.fnm) http://syslo.gs

Stored Fields

Stored fields are the original raw text values that were given to Lucene. This is not really part of the inverted index structure – its simply a mapping from document id's to stored field data. Stored fields are represented by two files:

1. The field index, or .fdx file. This contains, for each document, a pointer to its field data.

2. The field data, or .fdt file. This contains the stored fields of each document.



Term Dictionary (*.tis, *.tii)

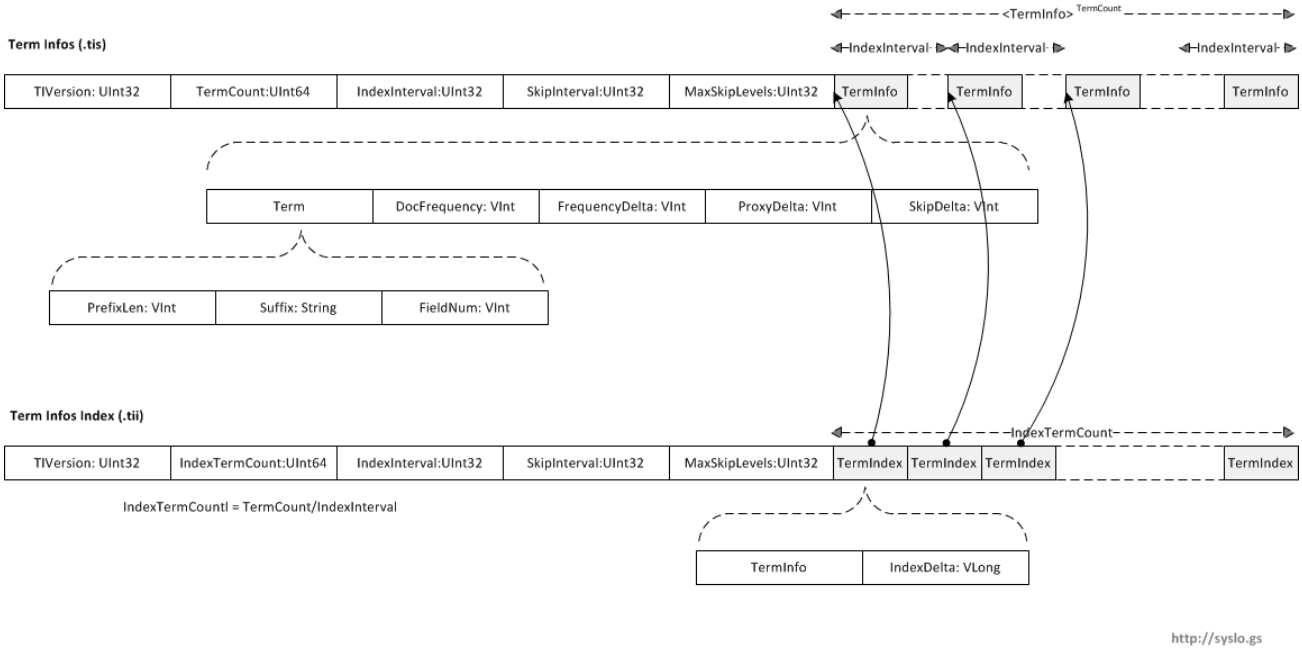
The Term Dictionary stores how to navigate the various other files for each term. At a simple, high level, the Term Dictionary will tell you where to look in the frq and prx files for further information related to that term. Terms are stored in alphabetical order for fast lookup. Further, another data structure, the Term Info Index, is designed to be read entirely into memory and used to provide random access to the "tis" file. Other book keeping (skip lists, index intervals) is also tracked with the Term Dictionary.

Term Infos (.tis)

Term Infos file (.tis) contains all terms in a segment ordered by field name then value within field. It also contain the document frequency for each term (Inverted index so how many document contain that term).

Term Info Index (.tii)

This contains every IndexIntervalth entry from the .tis file, along with its location in the "tis" file. This is designed to be read entirely into memory and used to provide random access to the "tis" file. The structure of this file is very similar to the .tis file, with the addition of one item per record, the IndexDelta.



IndexInterval: In order to accelerate the searching of a term, lucene uses a multi level skip-list. The index interval is the interval at which the terms are saved in the .tii file. For e.g., assuming IndexInterval=4, the dictionary index (tii) file will save the 4th, 8th, 12th term, which can speed up dictionary lookup in tis.

SkipInterval: SkipInterval is the fraction of TermDocs stored in skip tables. It is used to accelerate TermDocs.skipTo(int). Larger values result in smaller indexes, greater acceleration, but fewer accelerable cases, while smaller values result in bigger indexes, less acceleration (in case of a small value for MaxSkipLevels) and more accelerable cases.

MaxSkipLevels: MaxSkipLevels is the max. number of skip levels stored for each term in the .frq file. A low value results in smaller indexes but less acceleration, a larger value results in slighly larger indexes but greater acceleration. See format of .frq file for more information about skip levels.

PrefixLength: Term text prefixes are shared. The PrefixLength is the number of initial characters from the previous term which must be pre-pended to a term's suffix in order to form the term's text. Thus, if the previous term's text was "bone" and the term is "boy", the PrefixLength is two and the suffix is "y".

DocumentFrequency: DocFreq is the count of documents which contain the term.

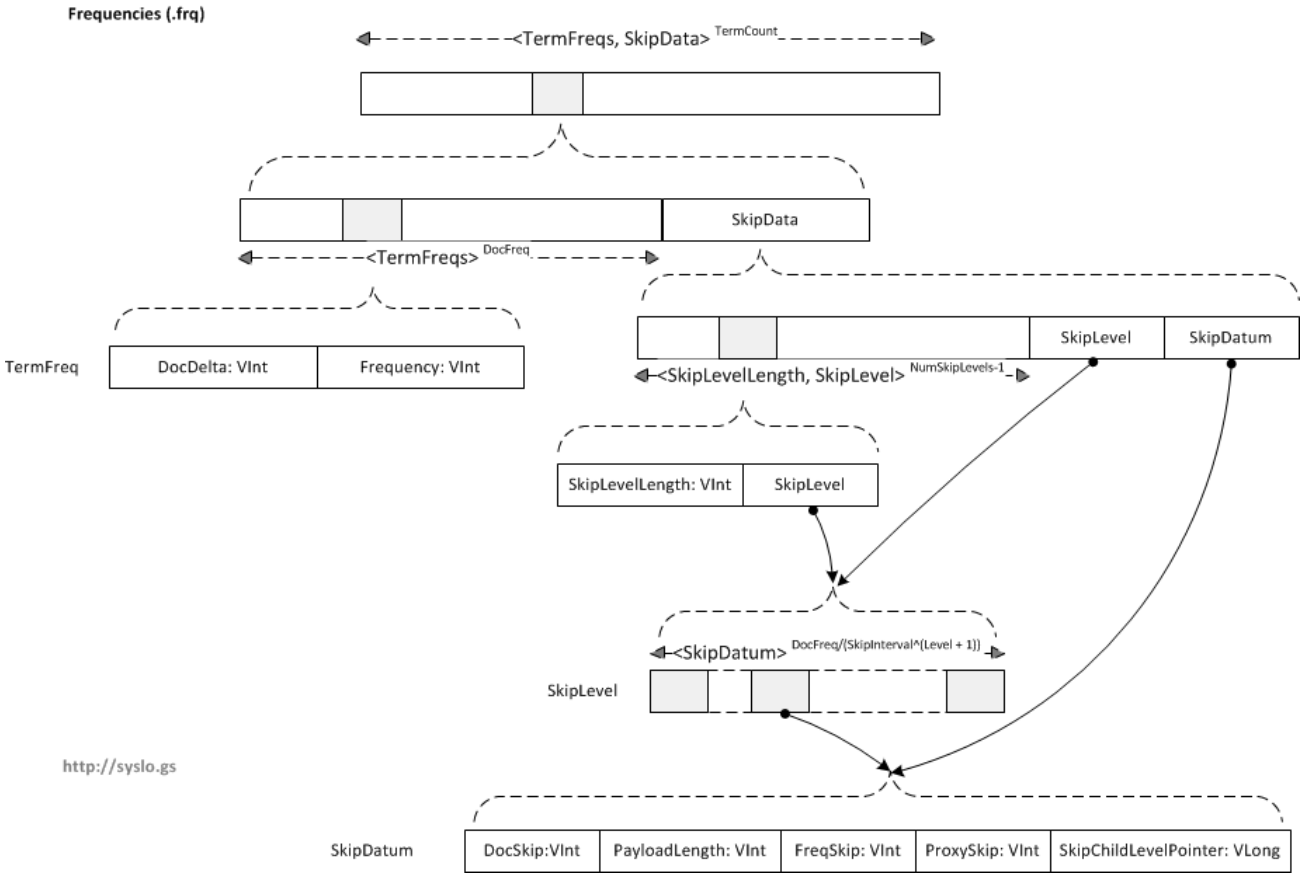
FrequencyDelta: FreqDelta determines the position of this term's TermFreqs within the .frq file. In particular, it is the difference between the position of this term's data in that file and the position of the previous term's data (or zero, for the first term in the file).

ProxyDelta: ProxDelta determines the position of this term's TermPositions within the .prx file. In particular, it is the difference between the position of this term's data in that file and the position of the previous term's data (or zero, for the first term in the file.

SkipDelta: SkipDelta determines the position of this term's SkipData within the .frq file. In particular, it is the number of bytes after TermFreqs that the SkipData starts. In other words, it is the length of the TermFreq data.

Term Frequencies (.frq)

The .frq file contains the lists of documents which contain each term, along with the frequency of the term in that document.



This file contains (TermPostingList * TermCount) entries. Each entry is a posting list corresponding to a term. Each posting list consists of Term Frequency and a Skip list.

Term Frequency consists of a list of DocDeltas, and the number of times this term occurs in that doc. The DocDelta encodes both the document number and the frequency. In particular, DocDelta/2 is the difference between this document number and the previous document number (or zero when this is the first document in a TermFreqs). When DocDelta is odd, the frequency is one. When DocDelta is even, the frequency is read as another VInt.

If omitTf = FALSE (Store the term frequency)

For example, if a term occurs once in document seven [DocID = 7, Freq = 1] and three times in document eleven [DocID = 11, Freq = 3], we would get the following sequence of VInts: 15, 8, 3. Here is how this is calculated:

In order to save space, Lucene does not store the doc id as is, instead it calculates a delta:

[DocIDDelta = 7, Freq = 1] [DocIDDelta = 4 (11-7), Freq = 3]

For the first term, since the frequency is 1, the frequency information is encoded in the DocDelta itself. DocIDDelta = 7 and in binary is 000 0111. Since we want to store the frequency of 1 in DocDelta, (7 << 1) | 1 = 15. The last bit in the docdelta is 1 (docid is odd), so we know that frequency is encoded in the docdelta and freq = 1.

For the second term, since frequency > 1, we use a separate vint to encode the frequency. DocIDDelta = 4 in binary 0000 0100. Since we want to show that the following int is for frequency and not another docid, we need to make the last bit 0 (zero). (4 << 1) | 0 = 8, then followed by real Freq = 3.

So to get the sequence: [DocDelta = 15] [DocDelta = 8, Freq = 3], namely sequence, 15,8,3.

If omitTf = TRUE (Skip storing term frequency)

If omitTf were true it would be this sequence of VInts instead: 7,4. If we omit term frequency, then all we are storing is the docdeltas. The DocDeltas are therefore [DocId=7][DocId=11] = [DocDelta=7] [DocDelta=11-7] = 7,4

SkipData: The skip data contains a set of skip lists that allows us to find a TermFrequency (DocDelta, freq) quickly.

- Number of skip levels in the SkipData = NumSkipLevels = Min (MaxSkipLevels, floor (log (DocFreq / log (SkipInterval)))). DocFreq = DocCount that contains this term.
- Number of nodes in each skip level = DocFreq / (SkipInterval ^ (Level + 1)), level >= 0
- All the SkipLevels are appended together, preceded by a SkipLevelLength. The last level does not need a SkipLevelLength since there are no more levels below it, so it is skipped.
- All SkipDatum in all levels, except the last level have a SkipChildLevelPtr which points to the next skip level.
- Each SkipDatum (SkipNode) contains the following information: document number, payload length, FreqSkip, ProxSkip, SkipChildLevelPtr.

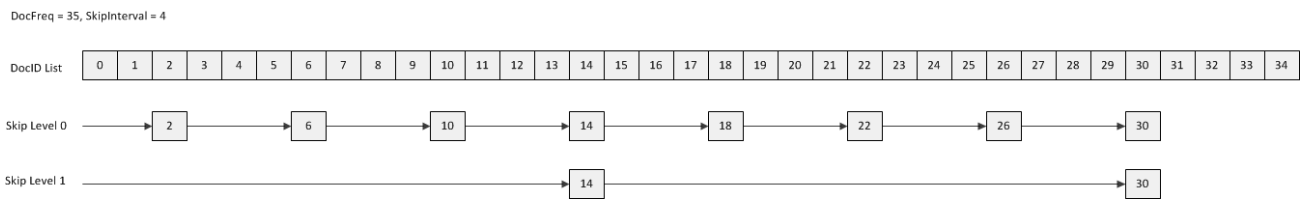
For Example:

If SkipInterval = 4, MaxSkipLevels = 2, DocFreq = 35, then:

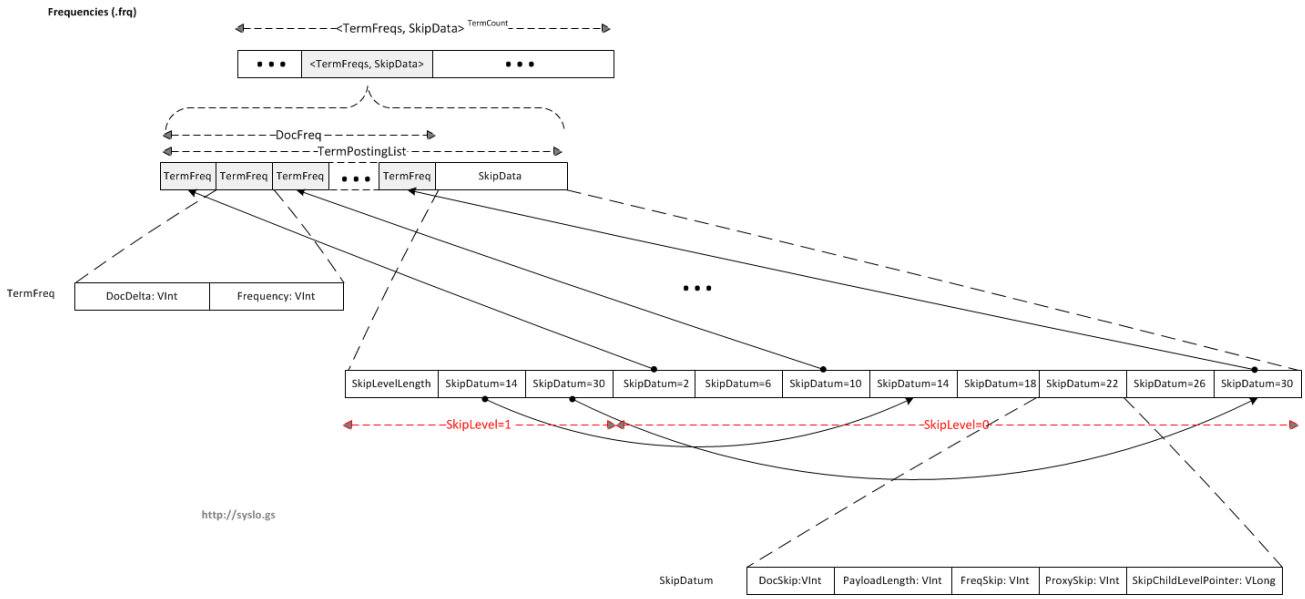
Skip level 0 has 8 SkipData entries (= 35 / (4 ^ (0 + 1))), containing the 3rd, 7th, 11th, 15th, 19th, 23rd, 27th, and 31st document numbers in TermFreqs.

Skip level 1 has 2 SkipData entries (= 35 / (4 ^ (1 + 1))), containing the 15th and 31st document numbers in TermFreqs.

Thus for SkipInterval 4, and when there are 35 documents, Skip level = 0 should include documents 3, 7, 11, 15, 19, 23, 27, and 31, and Skip level = 1 should include 15, 31 documents. But in the real implementation we store the previous docs as shown below.

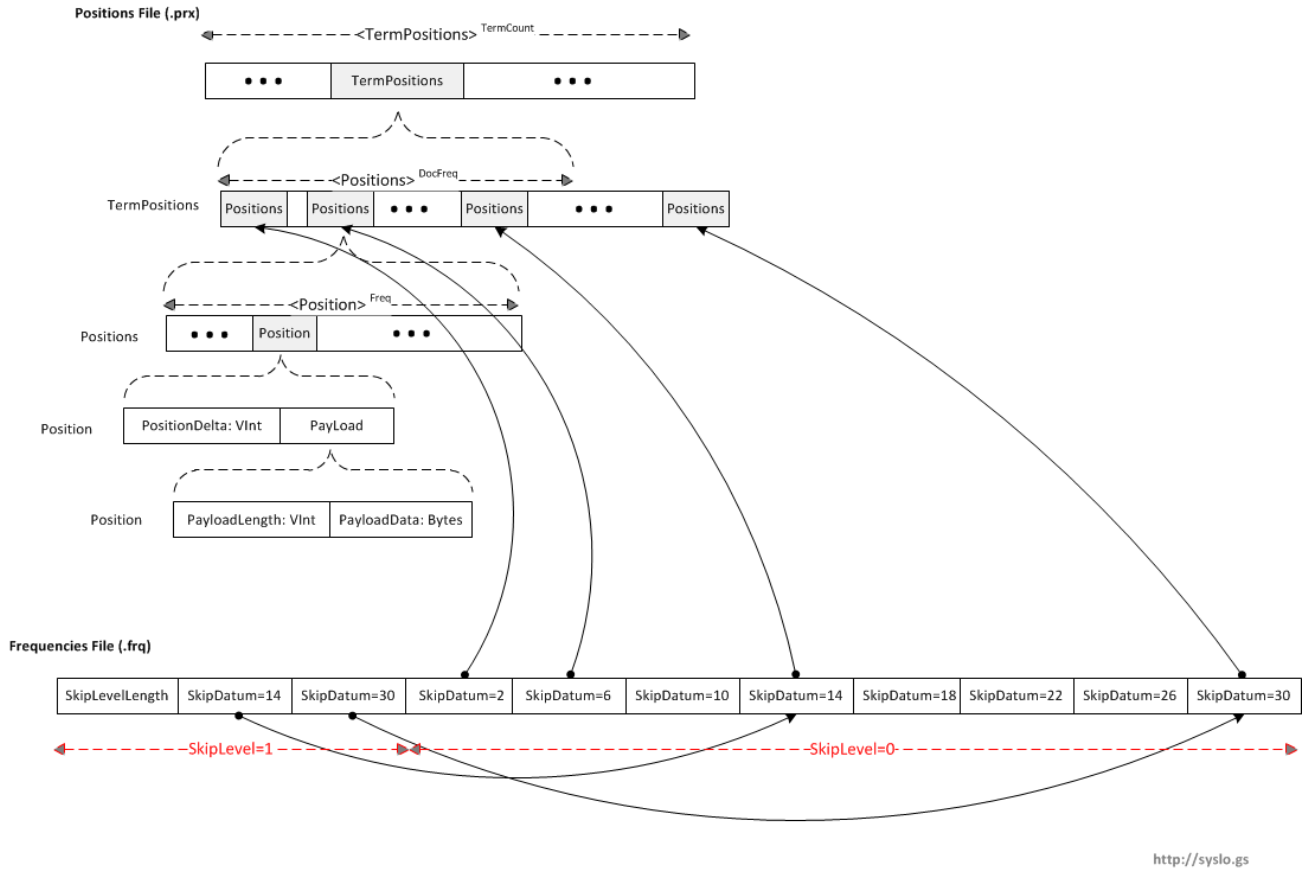


frequencies file with the above skip list:



Position Location Data(.prx)

The .prx file contains the lists of positions that each term occurs at within documents. If omitTf is set to true in a field, no entry for that term will be stored in this file, and if omitTf is set for all terms, the .prx files will not exist.



PositionDelta is the difference between the position of the current occurrence in the document and the previous occurrence (or zero, if this is the first occurrence in this document). For example, the TermPositions for a term which occurs as the fourth term in one document, and as the fifth and ninth term in a subsequent document, would be the following sequence of VInts: 4, 5, 4

Payload in Apache Lucene is an arbitrary byte array stored at a specific position (i.e. a specific token/term) in the index. A payload can be used to store weights for specific terms or things like part of

speech tags or other semantic information.

PayloadLength is the length of the PayloadData. If there is no payload length, but a PayloadData is found, its length is the same as the previous Payload.

PREVIOUS POST

← How big of a haystack do you need to hide?

NEXT POST

Dissicting the 82559 Ethernet controller – From bits to waves →

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Please prove you are not a bot

2 + 2 =

Post Comment