Index Compression - IR weekly report 4

hello underworld

6th June 2021

1 Some Advantages of Index Compression

1.1 In General

1. Use less disk space
2. Keep more stuff in memory
3. Increase speed of data transfer from disk to memory

1.2 For Inverted Index

For dictionary:
1).Make it small enough to keep in main memory
2).Make it so small that you can keep some postings lists in main memory too.
For postings lists:
1).Reduce disk space needed
2).Decrease time needed to read postings lists from disk
3).Large search engines keep a significant part of the postings in memory.

2 The Relationship between the collection and the vocabulary

Here, we discuss the two kinds of relationships between the collection and the corresponding vocabulary. One is the whole size of the vocabulary and the other is the each term frequency.

2.1 Heaps' Law

This model could estimates the number of terms in the whole collection. Here is the core function in this model:

$$M = k * T^b$$

where T is the number of tokens in the collection.Typical values for the parameters k and b are: $30 <= k <= 100$ and $b = 0.5$(largely).

Personally speaking, the parameters: k and b in this model should be calculated by the some input samples. And then, this model could be used to predict the size of the vocabulary of the similar input collection.

2.2 Zipf's law

This Zipf's law states that, if t1 is the most common term in the collection, t2 is the next most common, and so on, then the collection frequency cf_i of the ith most common term is proportional to $1/i$:

$$cf_i = k/i$$

Equivalently, we can write Zipf's law as $cf_i = ci^k$ or as $log(cf_i) = log(c) + k * log(i)$ where k = 1 and c is a constant. It is therefore a power law with exponent k = 1.

Actually, I still have some confusion about this model. To get the the rank of the term's occurrence, it should be done to calculate the term's frequency ahead. But maybe, this model get the frequency lists of each term through a small size of collection and then to predict the cf in the whole collection.

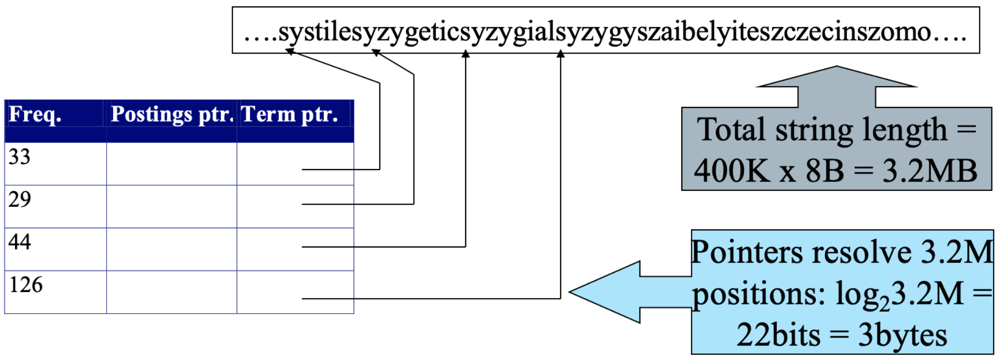
3 Some Dictionary Compression Schemes

After reading the relative parts of the book, we could kow the Fixed-Width terms are wasteful because the average size of English words is 8 charactes, with the 12 or bigger size of storge allocated in the Fixed-Width pattern. And with this pattern, there will be some problems if there are some rare terms which is really larger than the average size and can't be stored in the fixed-size pattern.
Here, we introduce a method: Dictionary as a string

3.1 Dictionary as a string

In this model, we treat the whole Dictionary as a string, allocating the continuous space for the terms. According to the size of the string, we can get the appropriate space for the pointer to the term in the string. Here is an example

of this model.
If there are about 400,000 terms and the average size of each term is 8 bytes. The term pointers resolve 400,000 $\times$ 8 = 3.2 $\times$ 106 positions, so they need to be $log_2(3.2)$ $\times$106 = 22bits or 3bytes long. In this new scheme, we need 400,000$\times$(4+4+3+8) = 7.6 MB for the dictionary: 4 bytes each for frequency and postings pointer, 3 bytes for the term pointer, and 8 bytes on average for the term.
Here is a simple example for this model:

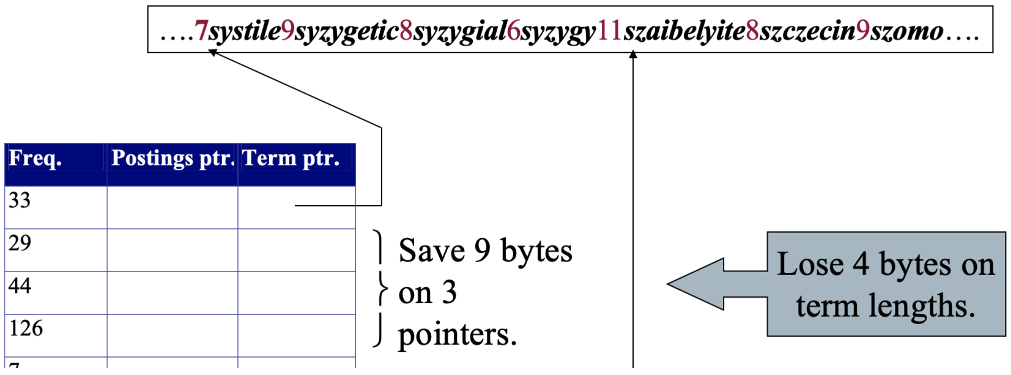


….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….
Freq.
Postings ptr.
Term ptr.
33
29
44
126
Total string length = 400K x 8B = 3.2MB
Pointers resolve 3.2M positions: log₂3.2M = 22bits = 3bytes

Figure 1: an example for Dictionary as a String

3.2 Blocked Storage

We can further compress the dictionary by grouping terms in the string into blocks of size k and keeping a term pointer only for the first term of each block. For example, if we set the size of block as 4(which means a block contains 4 terms).we used 3 bytes/pointer without blocking 3 x 4 = 12 bytes, now we use 3 + 4 = 7 bytes.



….7systile9syzygetic8syzygial6syzygy11szaibelyite8szczecin9szomo….
Freq.
Postings ptr.
Term ptr.
33
29
44
126
Save 9 bytes on 3 pointers.
Lose 4 bytes on term lengths.

Figure 2: an example for blocked storage, k = 4

However, there is a little disadvantage with the blocked storage. Due to the size of block, it will cost more time to retrieval the targeted term in the data structure. When k is the number of the whole terms, which means a block, you have to do a linear scan of whole terms, instead of a binary search in a binary tree of the dictionary.

3.3 Font coding

A sequence of terms with identical prefix is encoded by marking the end of the prefix with and replacing it with a diamond in subsequent terms. In this method, we could decrease the size of the string which contains the whole terms because some terms have the same part, prefix.Here is an example for this compression:

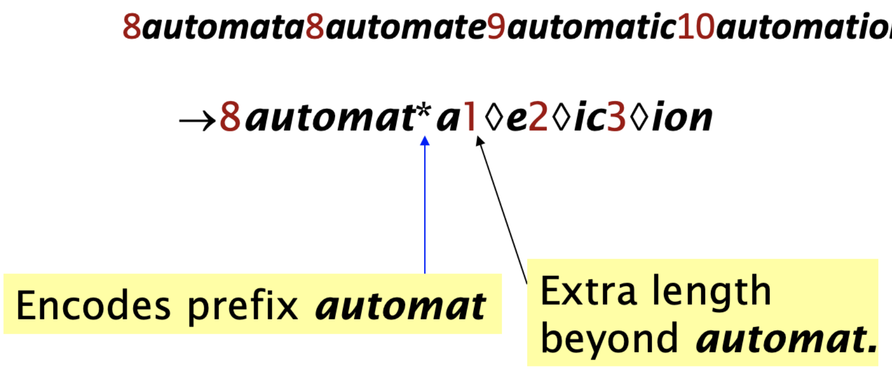

8automata8automate9automatic10automation
→8automat*a1◊e2◊ic3◊ion
Encodes prefix automat
Extra length beyond automat.

Figure 3: an example for font coding, prefix = 'automat'

4 Postings Compression

In the postings lists, there are several orders of magnitude data including the docID where the term occurs. Here are several methods for compressing it.

4.1 Gap encoding

Due to the huge size of the docID, the bits needed to encode the ID can be more and more. For example, when there are a million docID, 20bits are needed to encode a docID. However, we could store the gaps between two docID, instead of two indivisual docIDs. However, this will play roles when most the gaps can be stored with fewer bits. Here is an example for gap encoding.

	encoding	postings list								
the	docIDs	...		283042		283043		283044		283045
	gaps				1		1		1	
computer	docIDs	...		283047		283154		283159		283202
	gaps				107		5		43	
arachnocentric	docIDs	252000		500100						
	gaps	252000	248100							

Figure 4: an example for gap encoding

4.2 Variable Byte Code

Variable byte (VB) encoding uses an integral number of bytes to encode a gap. The last 7 bits of a byte are "payload" and encode part of the gap. The first bit of the byte is a continuation bit.It is set to 1 for the last byte of the encoded gap and to 0 otherwise. To decode a variable byte code, we read a sequence of bytes with continuation bit 0 terminated by a byte with continuation bit 1. We then extract and concatenate the 7-bit parts.
Here are some algorithms for this method:
The first one is to calculate the number of bytes needed:

```
VBEncodeNumber($n$)
1  $bytes \leftarrow \langle\rangle$
2  while $true$
3  do Prepend($bytes, n$ mod 128)
4      if $n < 128$
5         then Break
6      $n \leftarrow n$ div 128
7  $bytes$[Length($bytes$)] += 128
8  return $bytes$
```

Figure 5: the process of VBEN

The second one is to encode the gap:

```
VBEncode($numbers$)
1  $bytestream \leftarrow \langle\rangle$
2  for each $n \in numbers$
3  do $bytes \leftarrow$ VBEncodeNumber($n$)
4      $bytestream \leftarrow$ Extend($bytestream, bytes$)
5  return $bytestream$
```

Figure 6: the process of VBE

The third one is to decode the gap:

```
VBDecode(bytestream)
numbers ← ⟨⟩
n ← 0
for i ← 1 to Length(bytestream)
do if bytestream[i] < 128
      then n ← 128 × n + bytestream[i]
      else n ← 128 × n + (bytestream[i] − 128)
           Append(numbers, n)
           n ← 0
return numbers
```

Figure 7: the process of VBD

4.3 Gamma Coding

codes implement variable-length encoding by splitting the representation of a gap G into a pair of length and offset. Offset is G in binary, but with the leading 1 removed.2 For example, for 13 (binary 1101) offset is 101. Length encodes the length of offset in unary code. For 13, the length of offset is 3 bits, which is 1110 in unary. The code of 13 is therefore 1110101, the concatenation of length 1110 and offset 101.
Here are some propertys for Gamma coding:
1)G is encoded using 2 log G + 1 bits
2)Length of offset is floor(log G) bits
3)Length of length is floor(log G) + 1 bits
4)All gamma codes have an odd number of bits
5)Almost within a factor of 2 of best possible, log2 G
However, Gamma seldom used in practice.

5 Question

Actually, I still do not understand the usage of the Unary Code and the advantages of the method for coding.