Compression of the dictionary and posting lists
Summary of class discussions 2/13/11 thru 3/2/11

Remarks on Zipf's law (covered in Section 5.1.2 of *Introduction to Information Retrieval*):

General law: f_i = frequency of the i^{th} most frequent item = $i^{-\theta} f_1$

for some constant θ. For our application, items are terms that appear in the documents of a collection. One study gives θ of 1.5-2.0 for this application. The law is observed to hold for other applications with varying values of θ. The text *Introduction to Information Retrieval* focuses on $\theta = 1$. (The text also uses a general constant c rather than f_1.)

Taking logs, we have a linear relationship between $\log(f_i)$ and $\log(i)$:

$$\log (f_i) = \log(f_1) - \theta \log(i)$$

f_i could refer to either the fraction of the total number of occurrences or an actual count of occurrences. If f_i is the actual count of occurrences, *M* is the number of distinct terms and *T* is the total count of occurrences of all items, then

$$f_i = \frac{T}{\sum_{j=1}^{M} j^{-\theta}} i^{-\theta} .$$

($\sum_{j=1}^{M} j^{-\theta}$ is a well-known mathematical quantity: the order θ harmonic number of M.)

Heap's Law:
The material covered in class is identical to Section 5.1.1 of *Introduction to Information Retrieval*.

Dictionary compression:
The dictionary compression we considered in class is covered in Section 5.2 of *Introduction to Information Retrieval*.

I did a "back of the envelope" calculation in class to estimate the size of a modern Google dictionary from the size of the dictionary of the early Google in 1998:

Early Google: $n = 24 \times 10^6$ documents contain 14×10^6 unique terms.
Heaps law: dictionary size = $kn^{\beta} = k(24 \times 10^6)^{\beta} = 14 \times 10^6$

2010 Google: Google has reported that its new index structure is 100PB. Guessing that documents average 10KB of text gives us an estimate of 10 trillion documents.
For this $n = 10\times10^{12}$ documents, I used the rough expansion factor of 10^6 in document size from 1998 to 2010. Then:
new dictionary size = $k(24\times10^6 \times10^6)^{\beta} = k(24\times10^6)^{\beta} \times (10^6)^{\beta} = (14\times10^6) \times (10^6)^{\beta}$
Empirically, $\beta \approx 0.5$, which gives dictionary size $\approx (14\times10^6) \times (10^6)^{.5} \approx 14 \times10^9$
Thus we estimate a dictionary of 14 billion terms.

Using this estimate of dictionary size and using these values:

1 byte per character
8 characters *on average* per term
5-byte pointers into the character string of terms
8-byte pointers to the postings lists

compressing the dictionary using one long string of terms and pointers into the string requires approximately $(14\times10^9) \times (5 + 8 + 8) =$ 294 GB. Compare this to the $(14 \times10^9) \times (20 +8) =$ 392 GB of an array of term entries with 20 bytes allocated per term or to the $(14 \times10^9) \times (30 + 8) =$ 532 GB of an array with 30 bytes allocated per term. (Note that in class I ignored the space required for pointers to the postings lists since it is always the same.)

Posting-list compression:
We departed from the treatment in Section 5.3 of *Introduction to Information Retrieval* when we discussed bit-level variable-length codes for positive integers.

Notation:

1. string1 ◦ string2 denotes the concatenation of string1 and string2;
2. For any real number v, $\lfloor v \rfloor$ (read floor of v) denotes the largest integer less than or equal to v; for non-negative v, this is the same as the integer part of v.
3. For any real number v, $\lceil v \rceil$ (read ceiling of v) denotes the smallest integer greater than or equal to v.

Let x be a positive integer.

Unary representation of x: 11….10 with x 1's (same as in Section 5.3).

Elias γ-code for x:

unary rep. of $\lfloor \log x \rfloor$ ◦ $\lfloor \log x \rfloor$-bit binary rep. of $(x\text{-}2^{\lfloor \log x \rfloor})$

(Section 5.3 defines the same code from an alternate point of view, which you might find clearer. The alternate view does not illuminate the similarity to Golomb's code.)

In class, we did not explore what the encoding looks like specifically for powers of 2. We do that now:

x=1; $\lfloor \log 1 \rfloor = 0$.
We need the unary for 0 followed by the 0-bit binary representation of $1\text{-}2^0$: 0
x=2; $\lfloor \log 2 \rfloor = 1$.
We need the unary for 1 followed by the 1-bit binary representation of $2\text{-}2^1$: 100

$x=2^k$; $\lfloor \log x \rfloor = k$.
We need the unary for k followed by the k-bit binary representation of $2^k - 2^k$:
$\underbrace{1\ldots1}_{k}\ 0\ \underbrace{0\ldots0}_{k}$

Elias δ-code for x:

Elias γ-code for $\lfloor \log x \rfloor$ ◦ $\lfloor \log x \rfloor$-bit binary rep. of $(x\text{-}2^{\lfloor \log x \rfloor})$

The Elias γ-code for x is of length $2*\lfloor \log x \rfloor +1$, essentially twice the optimal length. The Elias δ-code for x is of length $2*\lfloor \log (\lfloor \log x \rfloor) \rfloor +1 + \lfloor \log x \rfloor$, which has an overhead in additional bits of essentially 2 times the log of the optimal length (i.e. 2loglogx) – a relatively small quantity for large x.

Example: 11100100000100011011010100101
1110 010 00001000011011010100101
Unary 3 give 2^3; add following 3-bit binary number 010 = 8+2 = 10 = $\lfloor \log x \rfloor$
111 0 010 0000100010 11010100101
2^{10} +10-bit binary number 0000100010 = 1024 + 34 = 1058 = x
The rest of the bits must represent a second number
110 10 100101
Unary 2 give 2^2; add following 2-bit binary number = 4+2 = 6 = $\lfloor \log y \rfloor$
110 10 100101
2^6 +6-bit binary number 100101 = 64 + 37 = 101 = y

I did a "back of the envelope" calculation in class to estimate the compression for the postings list of a term in a billion document collection if the fraction of the documents containing the term was 2^{-10} of the documents in the collection, or equivalently, $2^{30}*2^{-10}=2^{20}$ documents were on the postings list for the term. Making assumptions about the uniform distribution of the term among the documents, we expect gaps of average size 2^{10} between the IDs of consecutive documents in the postings list. We need 30 bits to represent all the document IDs, yielding $30*2^{20}$=30Mbits to list the document IDs in the postings list without compression. The Elias δ-code to represent gaps of size 2^{10} would take $2*\lfloor \log (\lfloor \log 2^{10} \rfloor) \rfloor +1 + \lfloor \log 2^{10} \rfloor = 2*3+1+10 = 17$ bits. Therefore representing 2^{20} gaps would take 17Mbits. (The extra bits to represent the full ID of the first document on the list are negligible.) This gives a 30:17 or almost 2:1 compression. Note that the smaller the gaps, the more we can save over the use of full 30-bit IDs for all the documents on the postings list.

Golomb code for x:

unary rep. of $\lfloor (x/b) \rfloor$ ◦ $\lceil \log b \rceil$-bit binary rep. of $(x - \lfloor (x/b) \rfloor * b)$

The Golomb code for x is of length $\lfloor (x/b) \rfloor + 1 + \lceil \log b \rceil$. This is a slightly simplified version of the Golomb code; the full version is one bit shorter in some instances. Quantity b is a parameter that must be chosen for each application. In the textbook *Modern Information Retrieval*, authors Baeza-Yates and Ribeiro-Neto claim that for compressing a sequence of gaps representing the postings list of documents for a term j, $b = 0.69(N/n_j)$ works well. N is the total number of documents, and n_j is the document frequency for term j (as used in tf-idf weighting for the vector model). The quantity N/n_j is an estimate of gap size. Note that b changes for each term in the lexicon, and all the documents must be processed to determine n_j before compressing the postings lists.

Compression numbers we looked at in class:

TREC-3 collection as compressed by Moffat and Zobel [†]:

2 GB of document data

Inverted index size without compression : 1.1 GB

Entries of the posting list for a term contain only (docID, term frequency in doc) pairs, not a list of occurrences within the document.

Compressed: 184 MB, a 6:1 compression

Gaps between document IDs in the posting lists are compressed used the Golomb code. (For this application, the Golomb code was shown to be slightly better than the Elias δ-code, which is better than the Elias γ-code.) The term frequency values are compressed using the Elias γ-code.

Reuters RCV1collection (more detailed numbers in Section 5.3.2 of *Introduction to Information Retrieval*.)

400MB postings lists uncompressed

116MB compressed by variable byte encoding.

101 MB compressed with Elias γ-code.

Skip pointers:

The basic idea of skip pointers can be found in Section 2.3 of *Introduction to Information Retrieval*. Our discussion added the use of gaps to represent documents in the chain of skip pointers. The original reference for all these ideas is the paper by Moffat and Zobel[†].

[†] A. Moffat and J. Zobel, Self-indexing inverted files for fast text retrieval, *ACM Transactions on Information Systems,* Vol. 14, No. 4 (Oct. 1996), pgs 349-379. Link provided on "Schedule and Assignments" Web page.