Compression of the dictionary and posting lists
Summary of class discussion – Part 2

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 ($x-2^{\lfloor \log x \rfloor}$)

(Section 5.3 defines the same code from an alternate point of view, which you might find clearer.)

Let us explore what the encoding looks like specifically for powers of 2:

x=1; $\lfloor \log 1 \rfloor = 0$.
We need the unary for 0 followed by the 0-bit binary representation of $1-2^0$: 0
x=2; $\lfloor \log 2 \rfloor = 1$.
We need the unary for 1 followed by the 1-bit binary representation of $2-2^1$: 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:
This replaces the unary in the Elias γ-code with Elias γ-code:

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

or equivalently

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

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: encoding 5000 = 4096+512+256+128+8
$\lfloor \log 5000 \rfloor = 12$, 5000 in binary is 1001110001000
Elias γ-code of (5000)=1111111111110001110001000
Elias δ-code of (5000)= 1110100001110001000

Example: decode 1110010000010001011010100101, encoded with Elias δ-code

1110 010 0000100010110101001 01

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

At this point we begin decoding 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

So the entire bit sequence represents x followed by y: 1058, 101

"Back of the envelope" calculation to estimate the compression for the postings list of a term in a 256 billion document collection if the fraction of the documents containing the term is 2^{-10} of the documents in the collection. Then $2^{38}*2^{-10}=2^{28}$ documents are on the postings list for the term (approximating 256 billion as 2^{38}). 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 38 bits to represent all the document IDs, yielding $38*2^{28}$ bits, or about 1 gigabyte, 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^{28} gaps would take $17* 2^{28}$ bits, or about 512 megabytes. (The extra bits to represent the full ID of the first document on the list are negligible.) This gives about a 2:1 compression. Note that the smaller the gaps, the more we can save over the use of full 38-bit IDs for all the documents on the postings list.

Golomb code for x:
The Golomb code is similar in structure to Elias γ-code.

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

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 (1994) as compressed by Moffat and Zobel ††:
1.7 million 1-KB documents for 1.7 GB of document data (document size enforced)
538,244 terms
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 (1996-1997) (see Section 5.3.2 of *Intro. to Info. Retrieval*.)
806,791 docs in ~ 1GB (~1.25 KB/doc)
391,623 terms
400MB postings lists uncompressed
116MB compressed by variable byte encoding, ~ 3.5:1 compression
101 MB compressed with Elias γ-code, ~ 4:1 compression

Compare more recent number, but unknown compression:

2004 Web crawl by Ntoulas & Cho (SIGIR 07)
130 million pages in 1.9TB (15KB/doc)
inverted index 1.2 TB

Google Caffeine 2011
index ~ 10PB

Skip pointers:
The basic idea of skip pointers can be found in Section 2.3 of *Introduction to Information Retrieval*. Our discussion adds 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[††].

Example:

Sequence of document IDs in a postings list:
5 8 12 13 15 18 23

Encoded using gaps:
5 +3 +4 +1 +2 +3 +5

add skip pointers to original list:
5 8 12 13 15 18 23
|_________↑ |_________↑

Encoded using one sequence of gaps for skips and sequences for gaps between skips:
5 +3 +4 +8 +2 +3 +10
|__________↑ |__________↑

To find a document one must follow the chain of skip pointers until the documents ID is found or until an ID (call it ID_{gtr}) greater than the desired one is reached, go back one pointer, and follow sequentially until the desired ID is found or ID_{gtr} is reached again, meaning the desired document is not present.

[†] Baeza -Yates, Ricardo and Ribeiro-Neto, Berthier, *Modern Information Retrieval*, Addison-Wesley, 1999.

[††] 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.