Short Problem Set (Module 3)

Note: when asked to calculate something, please show work; it often allows us to give partial credit for answers that are close by not entirely correct. In general you will not receive full credit for answers that are correct, but where there is no or inadequate justification provided.

1. [20%] Examine Table 5.1 in the text. The table reports the effect of various tokenization choices on the size of the index built from the Reuters RCV1 corpus.

(a) Expressed as a percentage of the original vocabulary size, how big is the vocabulary size (*i.e.,* the number of indexing terms) after removing numbers, case folding, stopword removal, and then stemming with the Porter stemmer?

(b) After removing numbers and case normalization, there are 96.97 million posting list entries in the inverted file. Removing 150 stop words reduces this to 67.00 million. If stemming is then performed, the number of posting lists entries reduces to 63.81 million. Explain **why** stemming makes a difference in the number of entries. A short example may be helpful.

(c) Is the mean posting list length longer or shorter after stemming is performed? Offer a brief explanation for the difference.

2. [30%] Express the numbers {32, 57, and 800} two ways: using a 12-bit binary representation and using gamma coding. You must follow the method for computing gamma described in the text and presented in the lecture materials. I strongly recommend learning to do this by hand, but you may write (and provide) a short computer program if you prefer – but do not use a program that you did not write yourself.

3. [20%] Below is a bit sequence for a gamma encoded gap list (as described in Chapter 5 of IIR and the lecture materials). Decode the gap list and reconstruct the corresponding list of docids. Spaces are added for ease of reading. Hint: there are six docids.

1110 0101 1110 1010 1001 1111 1000 0110 1101 1101

4. [30%] Researchers have proposed other schemes for gap list compression in inverted files. Some of these are slightly less space efficient, but may have faster implementations on modern hardware. Consider the following scheme called *Simple-9* (Anh & Moffat, "Inverted Index Compression Using Word-Aligned Binary Codes", 2005).

In *Simple-9* a 32-bit word is used to store between 1 and 28 numbers (gaps). The first four bits are used as a control to choose between nine options shown in the table below. The remaining 28 bits of each 32-bit word store numbers (gaps) using the same number of bits. For example, if the next 7 gaps are each between 1 and 16, then each of the 7 gaps will be represented in 4 bits. Note that since there are no gaps of size zero in a postings list, a gap of 1 is represented in 4 bits as "0000" and a gap of 16 is represented as "1111" -- in other words for *Simple-9* we store a gap, *g*, as the integer *g-1* in binary. The following table shows the control bits, the number of integers represented, the length of each code in bits, and the number of unused bits wasted at the end. In this problem, any leftover bits are set to 0.

4-bit control	# codes in 28-bit block	code length in bits	leftover bits	possible gaps
0000	28	1	0	1 or 2
0001	14	2	0	1 to 4
0010	9	3	1	1 to 8
0011	7	4	0	1 to 16
0100	5	5	3	1 to 32
0101	4	7	0	1 to 128
0110	3	9	1	1 to 512
0111	2	14	0	1 to 16384
1000	1	28	0	1 to 2^28

Using *Simple-9* encode a posting list for the following nine docids. Please write the bits four at a time, with a space between each four. Recall that you will first have to create a gap list before encoding gaps. Work carefully and do show your work. Docids: {4, 12, 15, 35, 36, 52, 102, 118, 218}