605.744: Information Retrieval
Problem Set (Module 3)

Sabbir Ahmed

September 19, 2022

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?

    Answer: The original vocabulary size was 484,494. The size after being reduced through the different levels of preprocessing is 322,383, which is 66.54% of the original vocabulary size.

    (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.

    Answer: Stemming terms leads to loss of information regardless of context. When terms are stemmed, depending on the algorithm used, the term may have its suffixes replaced with a simple fragment or have it entirely removed. Different words with similar prefixes of a term usually stem to identical terms, for example, both of the words "intern" and "internal" get stemmed to "intern" using a Porter stemmer. Although those words are unrelated, they get stemmed to the common term and the vocabulary size gets reduced by 1. Different conjugations of words also get reduced to their common stemmed word, such as "transparent" and "transparency" get stemmed to "transpar" using a Porter stemmer.

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

    Answer:

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.

    Answer: Gamma coding formula: $unary(floor(log_2(x))), binary(x - 2^{(floor(log_2(x)))})$

(a) 32

Answer:

- Converting to binary:

$2\,\underline{\big\vert\,32}$		Remainders
$2\,\underline{\big\vert\,16}$	—	0
$2\,\underline{\big\vert\,8}$	—	0
$2\,\underline{\big\vert\,4}$	—	0
$2\,\underline{\big\vert\,2}$	—	0
$2\,\underline{\big\vert\,0}$	—	0
1	—	1

  12-bit representation: 0000 0010 0000

- Converting to gamma coding:

$$\begin{aligned}
unary(floor(log_2(32))) &= unary(5)\\
&= 111110\\
binary(x - 2^{(floor(log_2(x)))}) &= binary(32 - 32)\\
&= 00000\\
gamma(32) &= 0111\ 1100\ 0000
\end{aligned}$$

(b) 57

Answer:

- Converting to binary:

$2\,\underline{\big\vert\,57}$		Remainders
$2\,\underline{\big\vert\,28}$	—	1
$2\,\underline{\big\vert\,14}$	—	0
$2\,\underline{\big\vert\,7}$	—	0
$2\,\underline{\big\vert\,3}$	—	1
$2\,\underline{\big\vert\,1}$	—	1
1	—	1

  12-bit representation: 0000 0011 1001

- Converting to gamma coding:

$$\begin{aligned}
unary(floor(log_2(57))) &= unary(5)\\
&= 111110\\
binary(x - 2^{(floor(log_2(x)))}) &= binary(57 - 32)\\
&= binary(25)\\
&= 11001\\
gamma(57) &= 0111\ 1101\ 1001
\end{aligned}$$

(c) 800

Answer:

- Converting to binary:

$$\begin{array}{r|lcc}
\multicolumn{2}{c}{} & & \text{Remainders} \\
2 & \underline{800} & & \\
2 & \underline{400} & — & 0 \\
2 & \underline{200} & — & 0 \\
2 & \underline{100} & — & 0 \\
2 & \underline{50} & — & 0 \\
2 & \underline{25} & — & 0 \\
2 & \underline{12} & — & 1 \\
2 & \underline{6} & — & 0 \\
2 & \underline{3} & — & 0 \\
2 & \underline{1} & — & 1 \\
\multicolumn{1}{r}{} & 1 & — & 1
\end{array}$$

  12-bit representation: 0011 0010 0000

- Converting to gamma coding:

$$\begin{aligned}
unary(floor(log_2(800))) &= unary(9) \\
&= 1111111110 \\
binary(x - 2^{(floor(log_2(x)))}) &= binary(800 - 512) \\
&= binary(288) \\
&= 100100000 \\
gamma(800) &= 0111\ 1111\ 1101\ 0010\ 0000
\end{aligned}$$

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

   Answer: Regrouping the bit sequence using the following algorithm:

   - Read the 1 bits until the first zero and store it as *magnitude* of the unary code
   - Read the *magnitude* number of bits as b = binary code
   - Compute $u = 2^{magnitude}$
   - Add u to the decimal value of the binary code, $u + dec(b)$

   Therefore,

   $$\begin{aligned}
   &\{1110010\ 111101010\ 100\ 1111110000110\ 11011\ 101\} \\
   &= \{2^3 + dec(010),\ 2^4 + dec(1010),\ 2^1 + dec(0),\ 2^6 + dec(000110),\ 2^2 + dec(11),\ 2^1 + dec(1)\} \\
   &= \{2^3 + 2,\ 2^4 + 10,\ 2^1 + 0,\ 2^6 + 6,\ 2^2 + 3,\ 2^1 + 1\} \\
   &= \{8 + 2,\ 16 + 10,\ 2 + 0,\ 64 + 6,\ 4 + 3,\ 2 + 1\} \\
   &= \{10,\ 26,\ 2,\ 70,\ 7,\ 3\}
   \end{aligned}$$

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}

Answer: Gap list of {4, 12, 15, 35, 36, 52, 102, 118, 218}:

{4, 8, 3, 20, 1, 16, 50, 16, 100}

Mapping the gaps to their range of possible gaps

gap	**range**	**4-bit control**
4	1 to 4	0001
8	1 to 8	0010
3	1 to 4	0001
20	1 to 32	0100
1	1 or 2	0000
16	1 to 16	0011
50	1 to 128	0101
16	1 to 16	0011
100	1 to 128	0101

Filling in the first 5 gaps, {4, 8, 3, 20, 1}, into the 0100 selector, which accepts gaps ranging 1 to 32:

4-bit control	**# codes in 28-bit block**	**code length in bits**	**gaps g**	**binary $g-1$**
0100	5	5	{4, 8, 3, 20, 1}	{00011, 00111, 00010, 10011, 00000}

Filling in the latter 4 gaps, {16, 50, 16, 100}, into the 0101 selector, which accepts gaps ranging 1 to 128:

4-bit control	**# codes in 28-bit block**	**code length in bits**	**gaps g**	**binary $g-1$**
0101	4	7	{16, 50, 16, 100}	{0001111, 0110001, 0001111, 1100011}

Combining the tables:

element	**4-bit control**	**gaps (decimal)**	**gaps (binary with leftover bits)**
0	0100	{4, 8, 3, 20, 1}	0001 1001 1100 0101 0011 0000 0000
1	0101	{16, 50, 16, 100}	0001 1110 1100 0100 0111 1110 0011