

Optimal encoding of monotonic integer sequences

Source: <http://sci.tech-archive.net/Archive/sci.math/2005-09/msg03804.html>

- *From:* "nightlight" <nightlight@xxxxxxxxxxxxxxxx>
 - *Date:* 20 Sep 2005 02:16:10 -0700
-

> From:

>

http://groups.google.com/group/comp.compression/browse_frm/thread/5e605caf477317f2/3dc7506159aef1f4?hl=en#3

>

> I have a long list of 80 bit integers stored in a 10 byte
> array. The data is sorted.... One obvious way of compressing
> this data is to store the differences....

>

> Am I missing some obvious far superior approach?

There are several improvements, depending on additional info you may have about the source of the numbers.

1) The most obvious improvement would apply to monotonically (de)increasing integer sequences which have additional regularity, such as linear or quadratic increase (or via any simple to compute monotonic function). To detect such regularity you could continue with taking differences of differences. For the linearly increasing sequence these second order differences will become zero. For quadratically increasing functions, the third order differences will become zero.... etc. Even if the higher order differences don't become exactly zero, they may become much smaller numbers than those in the original sequence, which would help the compression.

2) You may examine the statistics of the first (or higher) order differences. If the differences are not uniformly distributed, then they are compressable (e.g. via Huffman or Arithmetic coding). For example, for naturally occurring sequences, the small differences are often more frequent than the large differences. The case (1) is a special case of this type of regularity (when value 0 is highly overrepresented).

3) If none of the above applies (or after the regularities above have been extracted), you will have a sequence of integers uniformly filling the range from some minimum to some maximum value. This can always be remapped to a sequence from 0 to some max value M, again uniformly filling this range [0,M]. At this point storing the numbers separately as fixed length bitstrings will generally waste space, depending on the value M:

Optimal encoding of monotonic integer sequences

a) If M is an integer of the form $2^s - 1$ (such as 3, 7, 15, 31..) then the plain binary storage with fixed number of s bits per integer will be optimal.

b) If M is not of the form $2^s - 1$, then it always satisfies the inequality of the form:

$$A = 2^{(s-1)} - 1 < M < 2^s - 1 = B \dots (1)$$

Here you have two ways to proceed, one slightly suboptimal but quick and the other fully optimal but slower.

i) === Quick Suboptimal Method (slice codes) ===

Here you will encode integers not via the fixed length of s bits (which you could certainly do) but some integers via $(s-1)$ bits per integer and the rest via s bits per integer. Specifically, using the definitions of A and B from eq (1), you will encode the first $B-M$ integers (which is, by (1), always a positive number) using $(s-1)$ bits and encode $M+1-(B-M) = 2M-B+1$ remaining integers using s bits.

Example: let $M=5$, i.e. there are 6 distinct integers 0,1,2,3,4,5. Then the value B in (1) will be $7=2^3-1$ and value $A: 3=2^{(3-1)}-1$, and $s=3$. By the rule given above you would encode $B-M=7-5=2$ lowest integers 0 and 1 in $(s-1)=3-1=2$ bits and encode the $2M-B+1=10-7+1=4$ remaining integers in 3 bits. The encoding table is thus:

0	00
1	01
2	100
3	101
4	110
5	111

The plain encoding of method (a) uses 3 bits per integer, while method (i) uses $(2*2 + 4*3)/6 = 16/6 = 2.66\dots$ bits per integer, which is 89% of the plain encoding size. While clearly an improvement over the simple method (a), the theoretically optimal value is $\log_2(M+1) = \log_2(6) = 2.58$ bits per integer.

ii) === Optimal Method (radix codes) ===

You encode the integer sequence as a sequence of digits of some number X given in radix $R=M+1$. For example for $M=5$, you would encode integers as if they represent digits of a single number X in radix 6. If your n integers are D_1, D_2, \dots, D_n and for each D_i you have $0 \leq D_i < R$, then you compute X as:

Optimal encoding of monotonic integer sequences

$$X = D1 + R * D2 + R^2 * D3 + \dots + R^{(n-1)} * Dn \dots$$

(2)

or quicker via Horner scheme as:

$$X = ((\dots ((Dn * R + D[n-1]) * R + D[n-2]) * R + \dots + D2) * R + D1 \dots$$

(3)

Using (2) you can often store powers of R in a table. Note that both expressions (2) or (3) require arithmetic precision for multiplies and adds which is of the size (in bits) of the output, the number X. You can trade off some compression optimality for speed by breaking sequence of integers into fixed size blocks of just few integers, say Q integers, so that (2) or (3) are computed for each block separately and the arithmetic fits in, say 32 or 64 bits. To select good values of Q you can expand $\log_2(R)$ into continuous fractions and select Q from the denominators of 1st, 3rd, 5th... and other fractions of odd order.

For example for our earlier R=6 case, $L = \log_2(R) = 2.584962\dots$ which gives you continuous fractions approximating L as 3/1, 5/2, 13/5, 31/12, 106/41, 137/53, 791/306, ... etc. The first one 3/1=3 is the fixed 3 bit encoding of individual integers. The 3rd one 13/5 tells us to use blocks of 5 integers, in which case $X < 6^5 = 7776$ which uses 13 bits to represent X (encoding the 5 integers via eq (2)), thus yields $13/5 = 2.6$ bits per integer which is already better than the 2.666.. bits for method (i) and is nearly as quick. The next useful block size would be Q=41 integers (from 106/41). Here the $X < 6^{41}$ uses 106 bits for 41 integers, yielding $106/41 = 2.58536\dots$ bits per integer, which is a further improvement over the 2.6 bits we get for Q=5, but requiring now 106 bits of arithmetic precision. The next block size Q=306 from the fraction 791/306 yields encoding with $791/306 = 2.58490\dots$ bits per integer but requires 791 bits of arithmetic precision.

There is still a faster way than doing (2) and (3) directly, which is more optimal than the blocking. The method (invented by Boris Ryabko) is described in references [61] and [62], which are available at the web page:

<http://www.1stworks.com/ref/tr/tr05-0625r.htm>

There is finally an even faster and more optimal method than Ryabko's, called "Quantized Indexing" (QI) which will be published (along with the source code for encoder & decoder) at the site above in November this year. The QI is more general algorithm than the methods discussed above since it works as a general entropy coder and an optimal data structures (such as trees, graphs, self-delimiting sequences, etc) encoder. It is based on a recently discovered highly accelerated (about an order of magnitude faster than the fastest arithmetic coders) form of conventional enumerative/combinatorial coding (which is described in references 1, 11, 12, 13, 23 on the page above; the radix codes of eq's (2) and (3) above are a special case of the enumerative coding).

- *Follow-Ups:*
 - ◆ *Re: Optimal encoding of monotonic integer sequences*
 - ◇ *From:* nightlight
- Prev by Date: *Re: infinity*
- Next by Date: *Re: Sphere inside a pyramid*
- Previous by thread: *How to derive this step in the Ito integral?*
- Next by thread: *Re: Optimal encoding of monotonic integer sequences*
- Index(es):
 - ◆ *Date*
 - ◆ *Thread*