Gelasia binary number representation.

Francisco Casas B.

July 26, 2015

Abstract

The present document shows a way to represent integer numbers of any size on a binary digits array, saving at the same time both it's value and it's size, this in order to store or send a group of well delimited numbers on a compact and scalable way, ensuring the small numbers to use less space. This can also be used when seting up standars, avoiding issues related to indexation (like, the overflow of a fixed space to countain addresses, or a very expensive memory reserve for a large number range, that's not totally used on most of the cases).

Contents

1 Deduction

1.1 Previous analysis

Normally, the amount of numbers that's possible to represent on a given space of n bits is 2^n. If we consider only positive integers (because starting from 0 makes the deduction harder) the range of representable values is $[1, 2^n]$ then, on a rough approach, if it's necessary to represent a number and delimitate it, we would take the number on it's binary form (after substracting 1) and count the possition of the most significant 1 as the size of it, it's size has to be somehow stored, but that will be threated later, for now, the numbers will be represented this way:

```
 1 = [size:1] 0
 2 = [size:1] 1
 3 = [size:2] 10
 4 = [size:2] 11
 5 = [size:3] 100
 6 = [size:3] 101
 7 = [size:3] 110
 8 = [size:3] 111
 9 = [size:4] 1000
10 = [size:4] 1001
11 = [size:4] 1010
12 = [size:4] 1011
13 = [size:4] 1100
14 = [size:4] 1101
15 = [size:4] 1110
16 = [size:4] 1111
17 = [size:5] 10000
      ...
```

Then, if both the number and it's size are known, it's logical to think that if a number is big enough to requiere it's size, it's possible to dischard that the number has values that can be represented with smaller sizes:

```
 1 = [size:1] 0
 2 = [size:1] 1
 3 = [size:2] 00
 4 = [size:2] 01
 5 = [size:2] 10
 6 = [size:2] 11
 7 = [size:3] 000
 8 = [size:3] 001
 9 = [size:3] 010
10 = [size:3] 011
11 = [size:3] 100
12 = [size:3] 101
13 = [size:3] 110
14 = [size:3] 111
15 = [size:4] 0000
16 = [size:4] 0001
17 = [size:4] 0010
      ...
```

Now every possible value is being used, so, to get a number N given it's size S and this value X (that will be called extra from now), the next formula

will be used.

$$\begin{aligned} N &= X + 1 + \sum_{1 \leq i < S} 2^i \\ &= X + 1 + \frac{2^S - 2}{2 - 1} \\ &= X + 2^S - 1 \end{aligned}$$

Now, since the size S of a number, is also a number, it can be represented the same way, but, because $S > 0$, it's better to store $S - 1$. We can do so, until the size-1 of a size is 0, The result is as follows:

```
 1 = [size:1] 0
 2 = [size:1] 1
 3 = [size:2] 00 = [size:1] 0 00
 4 = [size:2] 01 = [size:1] 0 01
 5 = [size:2] 10 = [size:1] 0 10
 6 = [size:2] 11 = [size:1] 0 11
 7 = [size:3] 000 = [size:1] 1 000
 8 = [size:3] 001 = [size:1] 1 001
 9 = [size:3] 010 = [size:1] 1 010
10 = [size:3] 011 = [size:1] 1 011
11 = [size:3] 100 = [size:1] 1 100
12 = [size:3] 101 = [size:1] 1 101
13 = [size:3] 110 = [size:1] 1 110
14 = [size:3] 111 = [size:1] 1 111
15 = [size:4] 0000 = [size:2] 00 0000 = [size:1] 0 00 0000
16 = [size:4] 0001 = [size:2] 00 0001 = [size:1] 0 00 0001
17 = [size:4] 0010 = [size:2] 00 0010 = [size:1] 0 00 0010
     ...
```

The only information that's then needed to recover a number besides the secuence of digits on the right is how many times the size-1 representation algorithm was applied (this will be called recursivity), since this number grows on a very slow rate it can be represented on the most primitive way that allows an infinite size, this is a series of 1 ended by a 0, where the amount of 1's is the number of times that the algorithm had to be applied:

```
1 = 0 0
2 = 0 1
3 = 10 0 00
4 = 10 0 01
5 = 10 0 10
6 = 10 0 11
7 = 10 1 000
8 = 10 1 001
9 = 10 1 010
```

$$
\begin{aligned}
10 &= 10\ 1\ 011 \\
11 &= 10\ 1\ 100 \\
12 &= 10\ 1\ 101 \\
13 &= 10\ 1\ 110 \\
14 &= 10\ 1\ 111 \\
15 &= 110\ 0\ 00\ 0000 \\
16 &= 110\ 0\ 00\ 0001 \\
17 &= 110\ 0\ 00\ 0010 \\
18 &= 110\ 0\ 00\ 0011 \\
&\dots
\end{aligned}
$$

Not only the number it's stored on these bits, also it's size, that allows many numbers to be added contiguously on a bit array without need of a delimitator of additional data.

1.2 Unsigned gelasia representation

Now, with the last results, it's possible to substract 1 to the values in order to have a representation for the 0. This results on the following final representation:

$$
\begin{aligned}
0 &= 0\ 0 \\
1 &= 0\ 1 \\
2 &= 10\ 0\ 00 \\
3 &= 10\ 0\ 01 \\
4 &= 10\ 0\ 10 \\
5 &= 10\ 0\ 11 \\
6 &= 10\ 1\ 000 \\
7 &= 10\ 1\ 001 \\
8 &= 10\ 1\ 010 \\
9 &= 10\ 1\ 011 \\
10 &= 10\ 1\ 100 \\
11 &= 10\ 1\ 101 \\
12 &= 10\ 1\ 110 \\
13 &= 10\ 1\ 111 \\
14 &= 110\ 0\ 00\ 0000 \\
25 &= 110\ 0\ 00\ 1011 \\
125 &= 110\ 0\ 10\ 111111 \\
625 &= 110\ 1\ 001\ 001110011 \\
3125 &= 110\ 1\ 011\ 10000110111 \\
15625 &= 110\ 1\ 101\ 1110100001011 \\
78125 &= 1110\ 0\ 00\ 0000\ 0011000100101111 \\
390625 &= 1110\ 0\ 00\ 0010\ 011111010111100011 \\
1953125 &= 1110\ 0\ 00\ 0100\ 11011100110101100111 \\
9765625 &= 1110\ 0\ 00\ 0111\ 00101010000001011111011 \\
48828125 &= 1110\ 0\ 00\ 1001\ 011101001000011101011111 \\
244140625 &= 1110\ 0\ 00\ 1011\ 110100011010100101001010011 \\
1220703125 &= 1110\ 0\ 00\ 1110\ 001000110000100111001110010111
\end{aligned}
$$

```
  6103515625 = 1110  0  01  00000  01101011110011000100000111101011
 30517578125 = 1110  0  01  00010  11000110101111110101001001100001111
152587890625 = 1110  0  01  00101  00011100001101111001001101111111000011
                 ...
```

Where a number is determinated by the next formula, the calculation of the size S will be shown later:

$$N = X + 2^S - 2$$

1.3 Signed gelasia representation

It's also possible to add a sign bit when necessary, not substracting the 1 above mentioned to the absolute value of the negative numbers, in order to get only one representation of the value 0. This sign bit will be 1 for negative numbers and 0 otherwise:

```
        ...
−15 = 1  110  0  00  0000
−14 = 1  10  1  111
−13 = 1  10  1  110
−12 = 1  10  1  101
−11 = 1  10  1  100
−10 = 1  10  1  011
 −9 = 1  10  1  010
 −8 = 1  10  1  001
 −7 = 1  10  1  000
 −6 = 1  10  0  11
 −5 = 1  10  0  10
 −4 = 1  10  0  01
 −3 = 1  10  0  00
 −2 = 1  0  1
 −1 = 1  0  0
  0 = 0  0  0
  1 = 0  0  1
  2 = 0  10  0  00
  3 = 0  10  0  01
  4 = 0  10  0  10
  5 = 0  10  0  11
  6 = 0  10  1  000
  7 = 0  10  1  001
  8 = 0  10  1  010
  9 = 0  10  1  011
 10 = 0  10  1  100
 11 = 0  10  1  101
 12 = 0  10  1  110
 13 = 0  10  1  111
 14 = 0  110  0  00  0000
        ...
```

Then, if B is the sign bit of a number, it's value would be:

$$N = (X + 2^S - 2)(1 - 2B) - B$$

2 Calculation of a number's size

For a positive number N (before substracting 1) the size S is the position (from right to left, starting from 0) of the most significant 1 of the binary representation of the value $N + 1$, this also applies to a negative number with it's absolute value. And is deducted from the equation $N = X + 2^S - 1$. Once the value S is known, the following amounts of bits should be added to S to complete the total size.

S	recursivity	size indication	total
1	1	0	S+1
2-3	2	1	S+3
4-7	3	3	S+6
8-15	3	4	S+7
16-31	4	7	S+11
32-63	4	8	S+12
64-127	4	9	S+13
128-255	4	10	S+14
256-511	4	12	S+15
512-1023	4	13	S+16
1024-2047	4	14	S+17
2048-4095	4	15	S+18
4096-8191	4	16	S+19
8192-16383	4	17	S+20
16384-32767	4	18	S+21
32768-65535	4	19	S+22
65536-131071	5	23	S+28
131072-262143	5	24	S+29

Then, if an unsigned number is smaller than $2^{21} - 1 = 2097151$ (before substracting 1) it's granteed to use the same or less of the space that's possible to allocate on 32 bits.

If the representation is signed, 1 extra bit should be added.

3 Possible variantions

It's possible to set the minimal recursivity to 0 instead of 1, this will give a natural representation for the number 0 that only requires one bit, but will imply for all the other values to have an aditional 1 on it's recursivity header.

```
 0 = 0
 1 = 10  0
 2 = 10  1
 3 = 110  0  00
 4 = 110  0  01
 5 = 110  0  10
 6 = 110  0  11
 7 = 110  1  000
 8 = 110  1  001
 9 = 110  1  010
10 = 110  1  011
11 = 110  1  100
12 = 110  1  101
13 = 110  1  110
14 = 110  1  111
15 = 1110  0  00  0000
16 = 1110  0  00  0001
17 = 1110  0  00  0010
18 = 1110  0  00  0011
   ...
```

Also it's possible to add a base value to the size of all the numbers, for example 3:

```
 0 = 0  000
 1 = 0  001
 2 = 0  010
 3 = 0  011
 4 = 0  100
 5 = 0  101
 6 = 0  110
 7 = 0  111
 8 = 10  0  00  0000
 9 = 10  0  00  0001
24 = 10  0  01  00000
      ...
```

This modifications, among others, like applying an offset, can be made to ensure that the most frequent numbers use less space.