Home -> Gray Code

Gray code algorithms in Ruby

What is Gray code?

A Gray code is a way of encoding binary numbers so that only digit changes from one number to the next. That means that a single 0 can change to a 1, or a single 1 can change to a zero:

Decimal	Gray code
0	**0**000
1	000**1**
2	00**1**1
3	00**1**0
4	0**1**10
5	0**1**11
6	0**1**01
7	010**0**
8	**1**100
9	110**1**
10	11**1**1
11	111**0**
12	1**0**10
13	101**1**
14	10**0**1
15	100**0**

The highlighted digits are the ones that changed (counting up). The Gray code "wraps around:" When rolling over from 15 to 0, only one digit changes.

What is Gray code for?

Mechanical position sensors use Gray code to convert the angular position of a shaft to digital form. The shaft has a disk attached to it; the disk has tracks like a computer storage device. Some parts of a track have metal, corresponding to a "1", and other parts have insulator, corresponding to a "0". The sensor has a row of metal fingers, radiating out from the center of the disk, with one finger riding on each track. As the shaft rotates, the disk carries metal or insulator under each finger. The combination of 1's and 0's

read by the fingers indicate the angular position of the disk. Gray code is used instead of binary numbers because the fingers can't be lined up exactly. If two bits were to change at the same time (as in going from binary "01" to binary "10", one finger would invariably notice its bit change before the other finger did. That would be a very glitchy position sensor.

Generating Gray codes

A recursive algorithm

Here is a recursive algorithm for generating Gray codes:

1. Start with the Gray code sequence "0, 1"
2. Make a copy of the sequence.
3. Reverse the order of the copy.
4. Prepend a "0" to each number in original sequence.
5. Prepend a "1" to each number in the copy.
6. Concatenate the original and the copy together to create a new Gray code sequence twice as long as the original.
7. If you want more bits worth of Gray code, go back to step 2.

It's actually much simpler to look at when it's in code. So here is a Ruby example.

```
#!/usr/bin/env ruby

def prepend(prefix, array)
  array.collect { |item| prefix + item}
end

def grayCodes(bits)
  if bits == 1
    ["0", "1"]
  else
    prepend("0", grayCodes(bits - 1)) +
    prepend("1", grayCodes(bits - 1).reverse)
  end
end

puts grayCodes(4)
```

And here's the output:

```
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000
```

An iterative algorithm

Let's compare an ordinary binary sequence with Gray code. The left-most binary digit that changed is

highlighted. As before, the gray code bit that changed is highlighted.

Decimal	Gray code	Binary
0	**0**000	**0**000
1	000**1**	000**1**
2	00**1**1	00**1**0
3	00**1**0	00**1**1
4	0**1**10	0**1**00
5	011**1**	010**1**
6	01**0**1	01**1**0
7	010**0**	011**1**
8	**1**100	**1**000
9	110**1**	100**1**
10	11**1**1	10**1**0
11	111**0**	101**1**
12	1**0**10	1**1**00
13	101**1**	1101
14	10**0**1	11**1**0
15	100**0**	1111

Fascinating, isn't it? The digit to toggle when counting in Gray code happens to be the most significant digit that changed when counting in binary. Let's see if we can turn that into code.

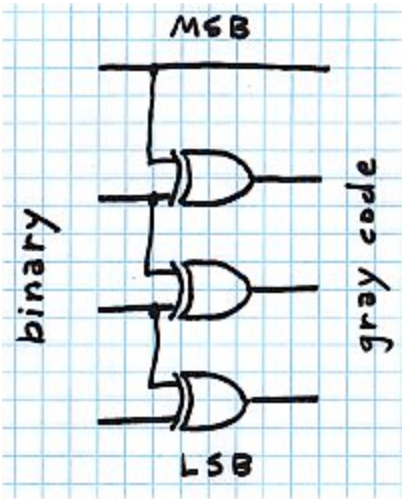
```
#!/usr/bin/env ruby

bits = 4
grayCode = 0
(1 << bits).times do |i|
  printf "%0#{bits}b\n", grayCode
  binaryBitsThatChanged = i ^ (i + 1)
  grayCodeBitToChange = binaryBitsThatChanged ^ (binaryBitsThatChanged >> 1)
  grayCode = grayCode ^ grayCodeBitToChange
end
```

Converting binary to Gray code

Here's the simplest algorithm for translating binary to Gray code. This algorithm can convert an arbitrary binary number to Gray code in finite time. Wonderful! Here it is in Ruby:

```
grayCode = binary ^ (binary >> 1)
```

How does it work? Beats me. It's a code version of this logic circuit using XOR gates (shown here for converting 4-bit numbers, but easily modified for any bit width):

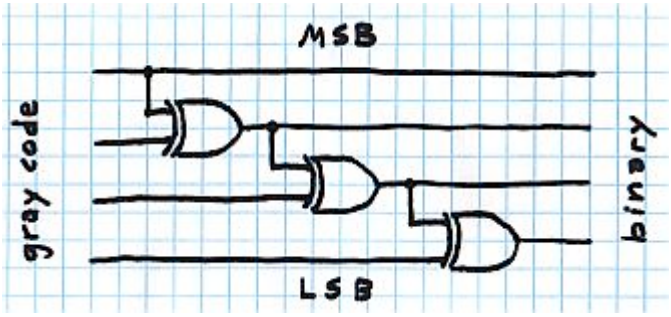


Converting Gray code to binary

It takes a little more to convert Gray code to binary. Here's the Ruby code:

```
# Convert binary to gray code.  bits is the number of bits to convert
# (could be calculated by looking at the size of the binary input).  binary
# is the number to convert.

def gray_code(bits, binary)
  gray_code = 0
  bit_number = bits - 1
  bit = 0
  while bit_number >= 0
    bit ^= binary >> bit_number & 1
    gray_code |= (1 << bit_number) * bit
    bit_number -= 1
  end
  gray_code
end
```

This code is easier to understand if you see the digital logic that it came from. It's just a cascade of XOR gates:



Every binary digit is the XOR of all gray code digits with equal or greater significance.

References

- Gray Code from The Free Online Dictionary of Computing.
- "What are Gray codes, and why are they used?" from The Hitch-Hiker's Guide to Evolutionary Computation.

- Gray Code from the NIST Dictionary of Algorithms, Data Structures, and Problems.
- Gray Codes from NUMERICAL RECIPES IN C.
