Fall 2012 Lab Assignment 6

You are to implement a method to compute the Gray code value for a sequence of integers.

A *Gray code*, named for Frank Gray, is a way to enumerate the bit patterns of a fixed length in such a way that the change from one pattern to the next only requires changing a single bit. Since enumerating the bit patterns of a given length n is also the same thing as listing the integers from 1 to 2^n, this is also one way to generate a bit mask for all possible subsets of a set of n items.

Consider the following tableau:

```
Counter, GrayCode   0: 0 0 0 0 :  0 0 0 0
Counter, GrayCode   1: 0 0 0 1 :  0 0 0 1
Counter, GrayCode   2: 0 0 1 0 :  0 0 1 1
Counter, GrayCode   3: 0 0 1 1 :  0 0 1 0
Counter, GrayCode   4: 0 1 0 0 :  0 1 1 0
Counter, GrayCode   5: 0 1 0 1 :  0 1 1 1
Counter, GrayCode   6: 0 1 1 0 :  0 1 0 1
Counter, GrayCode   7: 0 1 1 1 :  0 1 0 0
Counter, GrayCode   8: 1 0 0 0 :  1 1 0 0
Counter, GrayCode   9: 1 0 0 1 :  1 1 0 1
Counter, GrayCode  10: 1 0 1 0 :  1 1 1 1
Counter, GrayCode  11: 1 0 1 1 :  1 1 1 0
Counter, GrayCode  12: 1 1 0 0 :  1 0 1 0
Counter, GrayCode  13: 1 1 0 1 :  1 0 1 1
Counter, GrayCode  14: 1 1 1 0 :  1 0 0 1
Counter, GrayCode  15: 1 1 1 1 :  1 0 0 0
```

The first column of bits enumerates the integers from 0 through 16 in standard order. The second column of bits is a Gray code pattern for that integer. Note that going down the first column is simply adding 1 to each integer and then doing the schoolchild arithmetic to process the carries. The second column only has one bit changing from one row to the next.

This approach has been used extensively in hardware testing. It can also be used to generate all the subsets of a set of n elements. If one wanted to enumerate all 16 subsets of a set of 4 elements, the Gray code sequence above would allow one to do that in such a way as to only require adding or removing one element from each test subset.

(For what it's worth, "Gray code" refers to a sequence that enumerates and requires changing only one bit at a time. There are lots of way to produce a Gray code sequence, and we are only dealing with one such in this exercise.)

Note that the only code you need to produce here is the method for the Gray code itself.

One algorithm (and the one that I want you to implement) for generating the Gray code is this. If you write the integer k in binary, then you can start the Gray code algorithm by initializing the code to the binary pattern of the integer k. Now, as you process the bits of k from least significant (right hand end) to most significant, you invert the bit in the code if the next-higher bit in the integer is a 1.

Example:
For integer 13 = 1101 in binary, initialize the Gray code sequence to 1101. Now, look at the rightmost bit (the ones bit), which is a 1. The next bit to the left (the twos bit) is a 0, so we leave that bit in the Gray code alone and move left. Looking at the twos bit that is 0 in the representation for 13, we see that the next higher bit (the fours bit) is a 1, so we invert the zero, and the rightmost two bits of the Gray code value are now 11. Moving left again, the fours bit of 13 is a 1, and the eights bit of 13 is a 1, so we invert the bit in the Gray code, and the rightmost three bits of the Gray code are now 011.

Finally, since there is no higher bit beyond the eights bit, we always leave that one alone.

The resulting Gray code for 13 is 1011.

NOTE: My counter runs the subscripts backwards from standard notation, but then reverses them in the `toString` method. That is, subscript 0 is the ones bit, subscript 1 is the twos bit, and so forth.