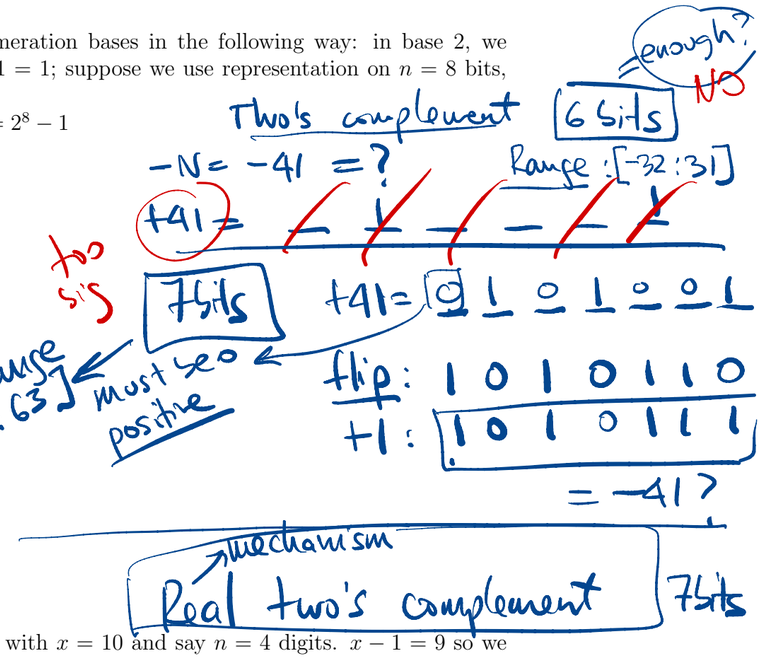
Geometric progression. For x real number and $n \geq 1$ integer we have
$(x^0 + x^1 + x^2 + ... + x^{n-1})(x - 1) = x^n - 1$
Exercise: prove this formula

This is relevant to numeration bases in the following way: in base 2, we make $x = 2$, thus $x - 1 = 1$; suppose we use representation on $n = 8$ bits, then
$2^0 + 2^1 + 2^2 + ... + 2^7 = 2^8 - 1$

or in binary
2^0 : 00000001 +
2^1 : 00000010 +
2^2 : 00000100 +
2^3 : 00001000 +
2^4 : 00010000 +
2^5 : 00100000 +
2^6 : 01000000 +
2^7 : 10000000

$2^8 - 1$: 11111111
1+

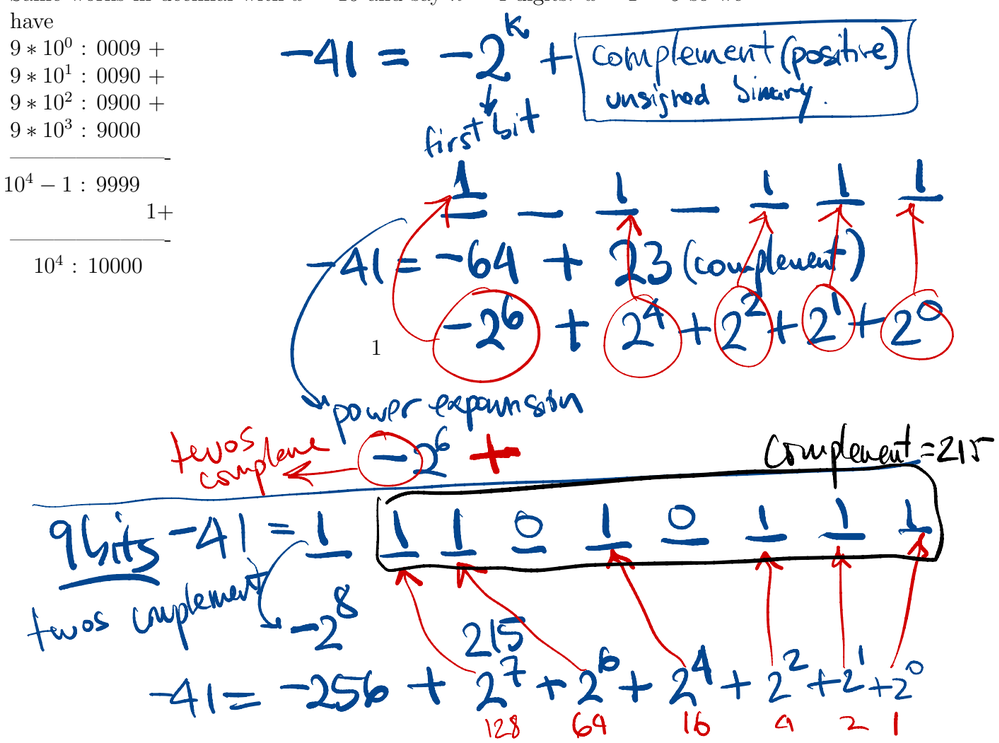
2^8 : 100000000

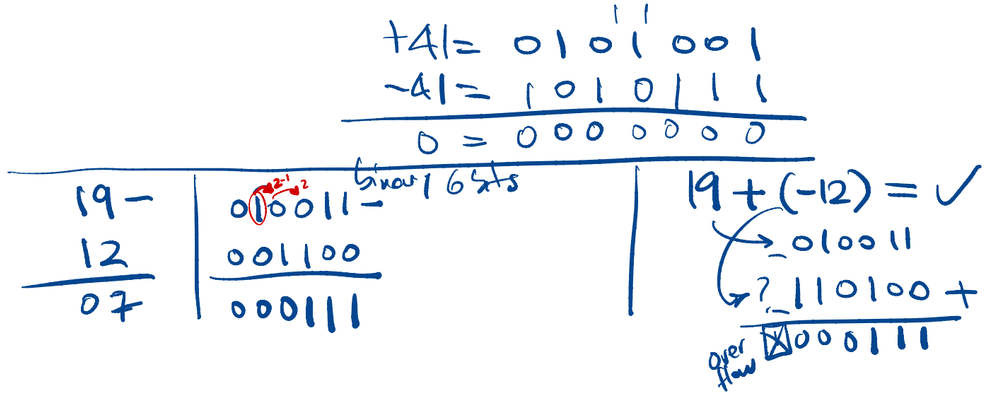


Same works in decimal with $x = 10$ and say $n = 4$ digits. $x - 1 = 9$ so we have
$9 * 10^0$: 0009 +
$9 * 10^1$: 0090 +
$9 * 10^2$: 0900 +
$9 * 10^3$: 9000

$10^4 - 1$: 9999
1+

10^4 : 10000

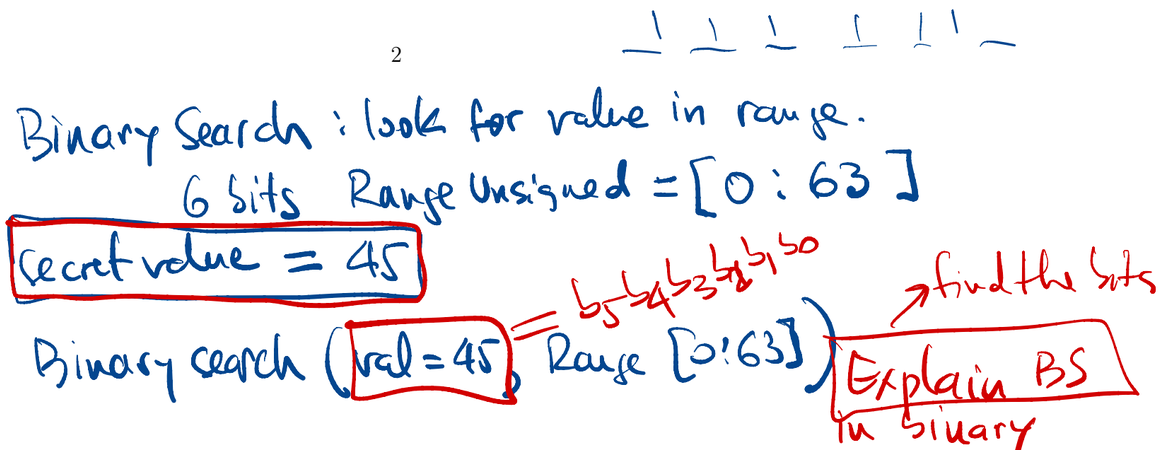


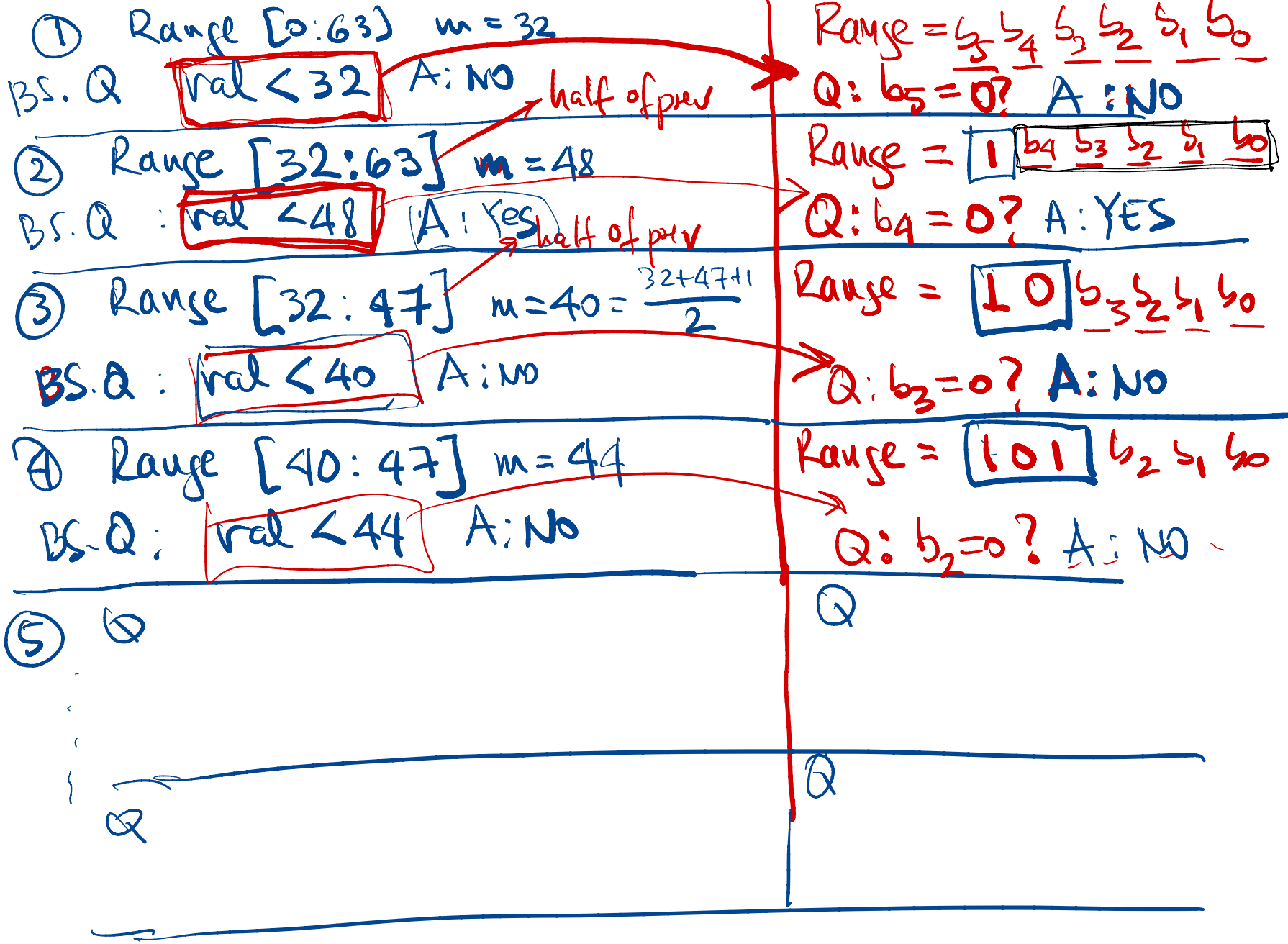


The maximum unsigned integer value in base x on n digits is when all digits are set to value $x-1$, that value is x^n-1. Adding 1 to it causes the need for another digit, the $n+1$ one, for the representation 10000..000 (1 followed by n zeros). If that $n+1$ digit is not available, we have an overflow problem: only the right-n digits are considered and the value is read as 0000..000 instead of x^n.

Exercise(geometric progression). Two players A and B play this game:
- A pays B \$1000 everyday for B-s lifetime
- B pays A for 30 days starting with 1 penny (\$0.01) the first day, doubling each day for the next 29 days; so that is 2 pennies the second day, 4 pennies in the third, 8 in the forth day etc.

Which player would you rather be?




① Range [0:63) m = 32
BS. Q val < 32 A: NO
② Range [32:63] m = 48
BS. Q : val < 48 A: Yes
③ Range [32:47] m = 40 = 32+47+1 / 2
BS. Q : val < 40 A: NO
④ Range [40:47] m = 44
BS. Q : val < 44 A: NO
⑤
half of prev
half of prev
Range = b5 b4 b3 b2 b1 b0
Q: b5 = 0? A: NO
Range = 1 b4 b3 b2 b1 b0
Q: b4 = 0? A: YES
Range = 10 b3 b2 b1 b0
Q: b3 = 0? A: NO
Range = 101 b2 b1 b0
Q: b2 = 0? A: NO
Q
Q
Q

Bitwise Operations

Shift left "<<". For integers, same as multiply by 2 for each bit shifted. Move all bits left by k positions, add k zeros to the right. Significant bits can be lost on the left size, still on same n bits. In the example below we represent an integer a=79 on $n = 32$ bits

$$\begin{aligned} 79 &= 0000\ 0000\ 0100\ 1111 \\ 79 << 1 &= 0000\ 0000\ 1001\ 1110 \\ &= 158 \\ 79 << 2 &= 0000\ 0001\ 0011\ 1100 \\ &= 316 \end{aligned}$$

Math
shift to left by 1 pos | ×2
shift to left by 2 pos | ×4

Shift right ">>". For integers, same as division by 2 for each bit shifted. Move all bits right by k positions, add k zeros to the left. Non-significant bits will be lost on the right size, as result is still on same n bits.

$$\begin{aligned} 79 &= 0000\ 0000\ 0100\ 1111 \\ 79 >> 1 &= 0000\ 0000\ 0010\ 0111 \\ &= 39 \\ 79 >> 2 &= 0000\ 0000\ 0001\ 0011 \\ &= 19 \end{aligned}$$

Math
shift to right by 1 pos | ÷2
shift to the right by 2 pos | ÷4

Bitwise AND "&". Given an integer mask m on 32 bits, the operation $y = m\&x$ performs a bitwise AND: all 0 bits in m produce 0 bits in y, while all 1-bits in m simply leave the corresponding bit in x to pass to y. For

example x=78, $m = 5$ gives y as:

$$\begin{aligned} x = 78 &= 0000\ 0000\ 0100\ 1110 \\ m = 5 &= 0000\ 0000\ 0000\ 0101 \quad \& \\ \hline y &= 0000\ 0000\ 0000\ 0100 \\ &= 4 \end{aligned}$$

This is particularly useful when $m = 2^k$ (a power of two), in order to check if the k-bit in x is one or zero:
if $2^k \& x == 0$ then k-th bit in x is 0; otherwise the x k-th bit is one.

Bitwise OR "|". Given an integer mask m on 32 bits, the operation $y = m|x$ performs a bitwise OR: all 1-bits in m produce 1-bits in y, while all 0-bits in m simply leave the corresponding bit in x to pass to y. For example x=78, $m = 21$ gives y as:

$$\begin{aligned} x = 78 &= 0000\ 0000\ 0100\ 1110 \\ m = 21 &= 0000\ 0000\ 0001\ 0101 \quad | \\ \hline y &= 0000\ 0000\ 0101\ 1111 \\ &= 95 \end{aligned}$$

This is particularly useful when $m = 2^k$ (a power of two), in order to make the k-bit in x one:
if $y = 2^k|x$ makes the k-th bit in y one, but leaves all other bits as in x.

Exercise. Play with the attached C code "bitwise.cpp". You dont have to look into declarations of variables, but rather change the integer values and see what happens. Being C++ code, you will have to compile and run it; you can do so with the attached Makefile, on a UNIX-based system, by simple typing in the terminal window
`make FILE=bitwise`
which will both compile and run the code. Every edit of the source code have to be saved and followed by the same `make` command.

```
19:13>> make FILE=bitwise
g++ -Wall -pedantic -o bitwise bitwise.cpp
./bitwise
size_of_int=4
a11=150000 a12=150000 OVERFLOW (32 bits)?
a11*a12=1025163520

a14=79
a15=a14>>1=39
a16=a14<<1=316

a1=79
11110010 00000000 00000000 00000000
a1=39
11100100 00000000 00000000 00000000
a1=316
00111100 10000000 00000000 00000000


a2=316

d1=10234.9
00011101 01111000 10100001 11010111 10001110 10111111 11000011 00000010
result=00011101011110001010000111010111100011101011111111000011000000010
d2=10234.9
```

Representation range on integers on 8 bits

Like before $x = 2, n = 8$ means 256 possibilities.

* unsigned (positives only) range is 0:255

0: 00000000 (min)
1: 00000001
2: 00000010
. . . . . .
255: 11111111 (max)

* signed (positives and negatives) range is -128:+127.

Two-complement. To represent negative integers we need one bit for the sign, so the magnitude of numbers is cut in half because now we represent both negative and positives. Positives have the first bit 0 and the regular binary representation; negatives $-v$ are represented on 8 bits as $2^8 - v$; which is at least $2^8 - 128 = 128$ on 8 bits, so all of the negatives have the first bit 1:

-128: 10000000 (min)
-127: 10000001
-126: 10000010
-125: 10000011
. . . . . .
-2: 11111110
-3: 11111101
-1: 11111111
0: 00000000
1: 00000001
2: 00000010
3: 00000011
. . . . . .
125: 01111101
126: 01111110
127: 01111111 (max)

Exercise: Verify that for any value v in range, in binary representation above $v + -v = 2^8$, or 10000000 which of course is 00000000 if we consider only 8 bits.

Exercise: Verify that the following procedure is equivalent for obtaining negative representation of value $-v$ in range:
* 1) get representation of positive v
* 2) flip all bits
* 3) add 1

Exercise. Verify that addition and subtraction works correctly with negative integers (two complement) on 8 bits. The result first bit might be incorrect (invalid) if there is an overflow. For example, verify that
6 - 7 = -1
-4 + -5 =-9
101 - 117 = -16
etc

Number of independent possibilities. If one has three jackets, 2 pants, and 4 hats, and *any combination is valid*, then how many combinations we have?
Answer: 3 x 2 x 4 = 24 possibilities

If number are represented in base 5 (thus digit values 0:4) on 8 digits, and any combination is allowed (like starting with 0), how many possible representations are there?
Answer: 5 (first digit) x 5(second digit) x ... x 5(eighth digit) = 5^8 possibilities

If numbers are represented in base 2 on 8 bits, and the first bit is for sign, what is the range of absolute values represented?
Answer: since we only have 7 bits for the actual value representation, there are $2^7 = 128$ possibilities. Centering a range at 0, positives will be 0:127 and negatives -128:-1.

If a set has 6 elements $S = \{e_1, e_2, ..., e_6\}$ how many different subsets are there? A subset is any set of made of elements in S, for example $\{e_2, e_3, e_5\}$; the empty set $\emptyset$ counts as one subset. The order of elements doesnt matter in a set, so $\{e_2, e_3, e_5\} = \{e_5, e_2, e_3\}$
Answer: there are 2 possibilities for each element in a subset: to be in, or to be out. So there are 2 possibilites for e_1, 2 for e_2, ... , 2 for e_6. Total is 2^6
Exercise. What if we count the ordered sets as different, i.e. $\{e_2, e_3, e_5\} \neq \{e_5, e_2, e_3\}$. How many possibilities we have now? Hint: many more than 2^6

The card trick presented at first lecture asked you to pick a number in range 0:63 and return precisely the subset of cards that contain it. To determine the number every time without error, the number of possible feedbacks (the possible subsets of cards returned) has to be at least 64, which is the number of possibilites for the number. If thats not true, simply there wouldn't be enough ways to distinguish 64 numbers. Where there at least 64 feedback responses?
Answer: yes. There were 6 cards, so 2^6 possibilities to create a subset of cards returned.

How many bits are necessary to represent all the positive range 0:1200?
Answer: 1201 possibilities, ignoring the sign, cant be represented on 10 bits

because that gives only $2^{10} = 1024$ possibilities; but 11 bits is enough since that gives $2^{11} = 2048$ possibilities.

Exercise. The explanation for the card trick (pdf linked form the website) is described with 24 cards in base 10 covering numbers in range 0:350. Lets say range 0:350 is fixed, but you can pick any numeration base b (for example if you pick $b = 16$ you'll have to write the numbers in hex with digits 0:F). What base minimizes the number of cards needed?Hint: in base b the number of cards will be about `#bits` $\times$ `#cards-per-bit`

If there are 10 adjacent houses in a row to be painted either Red, Blue, Green, Yellow and *any coloring is possible*, how many total colorings are there?
Answer: 4 for first house x 4 for second x etc, so 4^{10} total.
Exercise. What if we require that any two adjacent houses have different colors?

Exercise(difficulty ★). What if we require that any sequence of three consecutive houses must have have different colors (cannot have same color for any 2 houses within three consecutive)?

Exercise (difficulty ★★) What if have specific requirements: In a given order (say close to far) Red color can be followed by any other color; Blue can be followed only by Green and Yellow; Green can be followed by any color except Red. How many total colorings now?