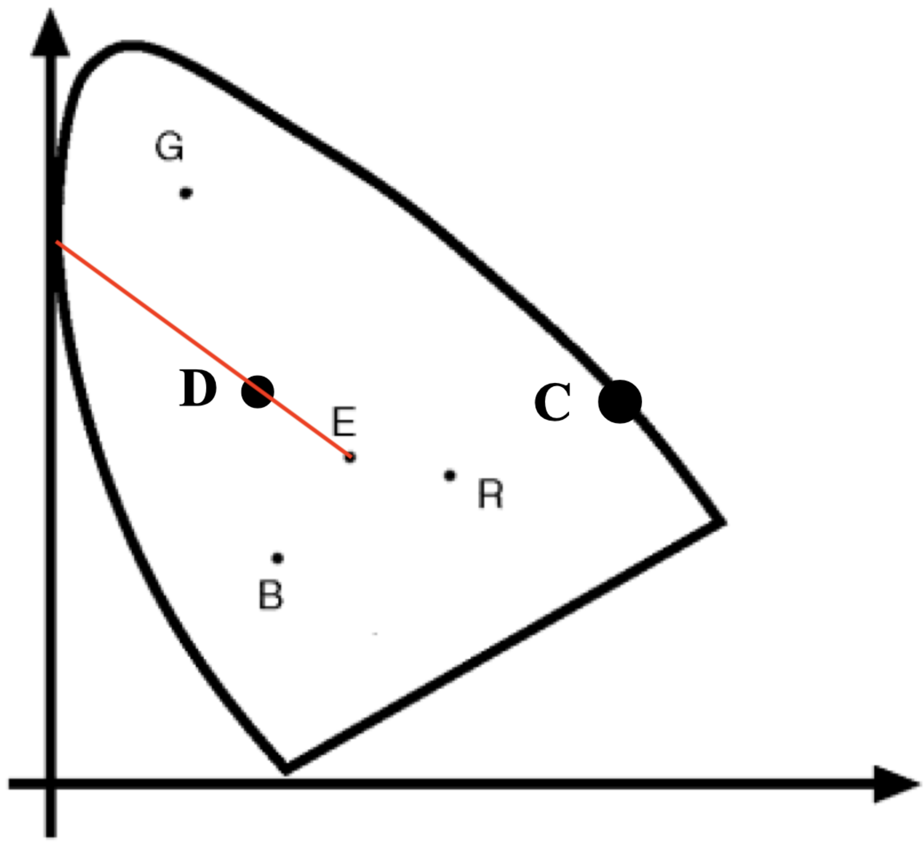
Question 1: Color Theory

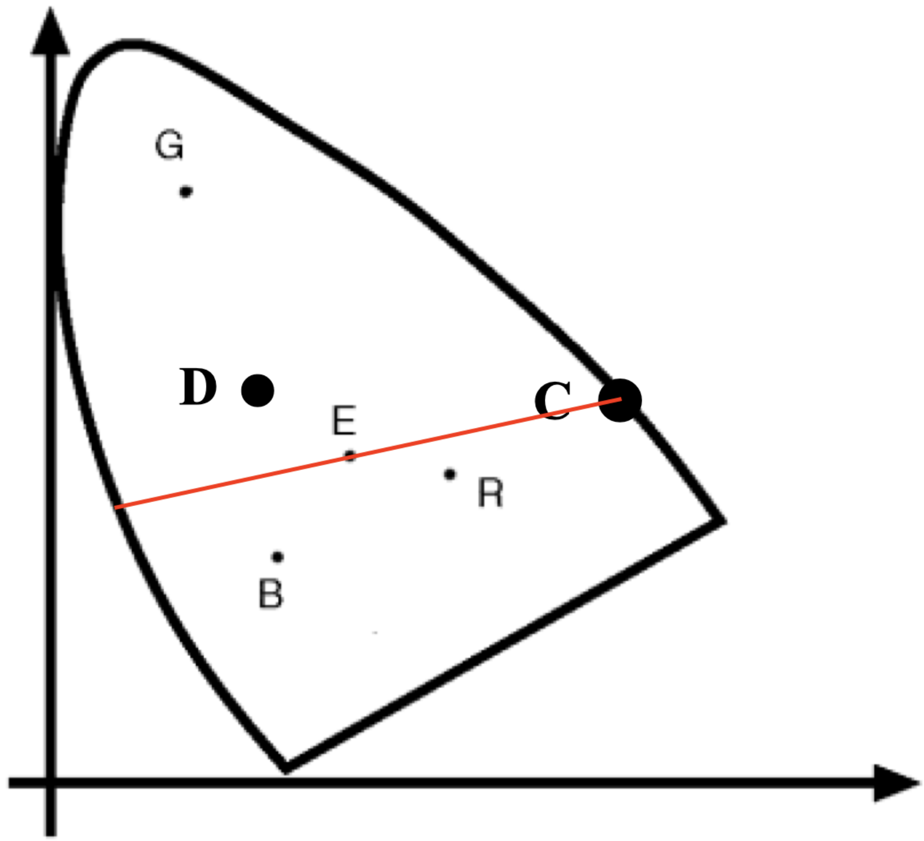
(1) Dominant wavelength is found by drawing a line from E passing D that projects onto the curve corresponding to the spectrum.



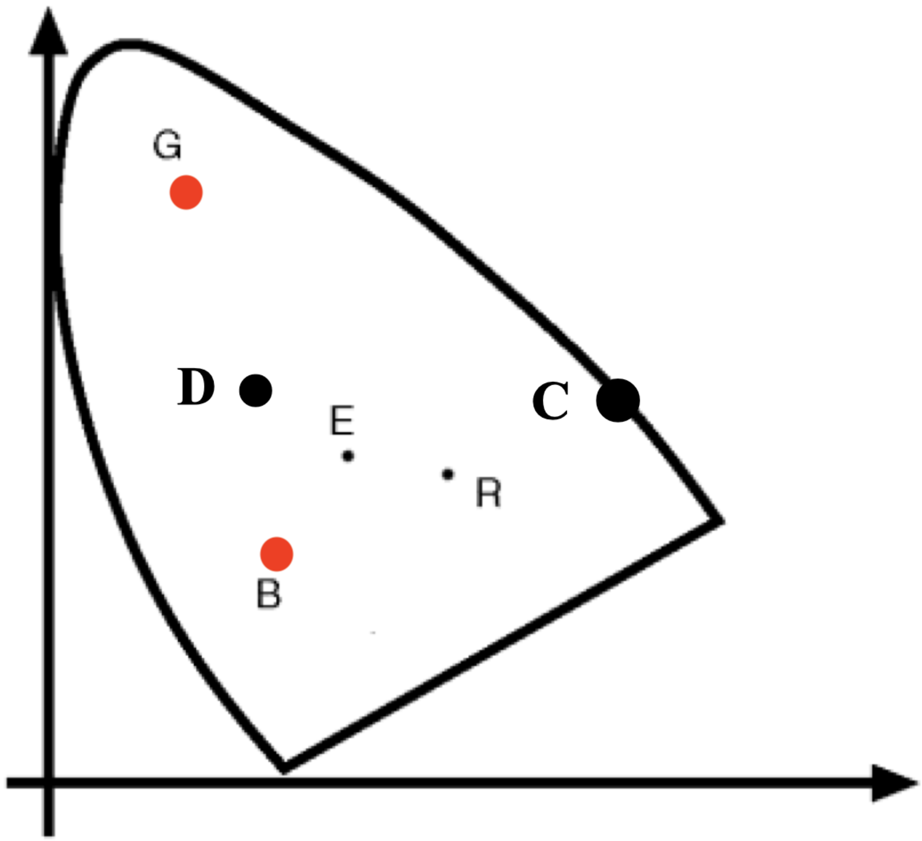
(2) No; the flat line at the bottom correspond to non-spectrum colors, so colors whose lines from E project to this flat line don't have dominant wavelength.

(3)

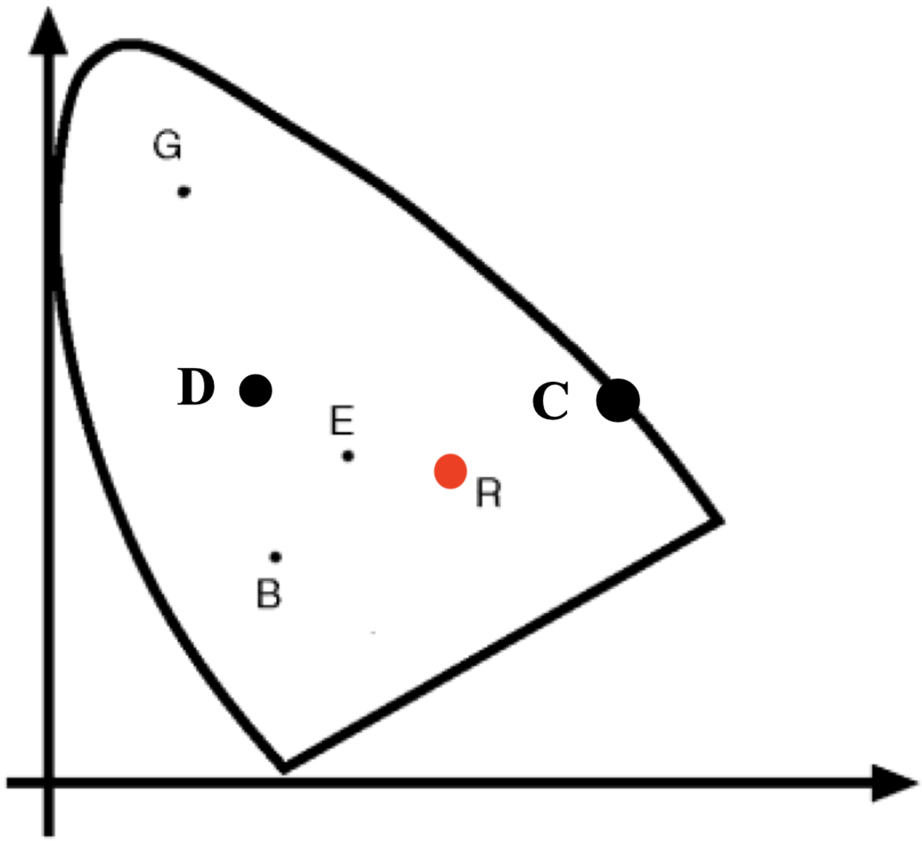
Complement can be found by drawing a line from C passing E that projects onto the curve corresponding to the spectrum.



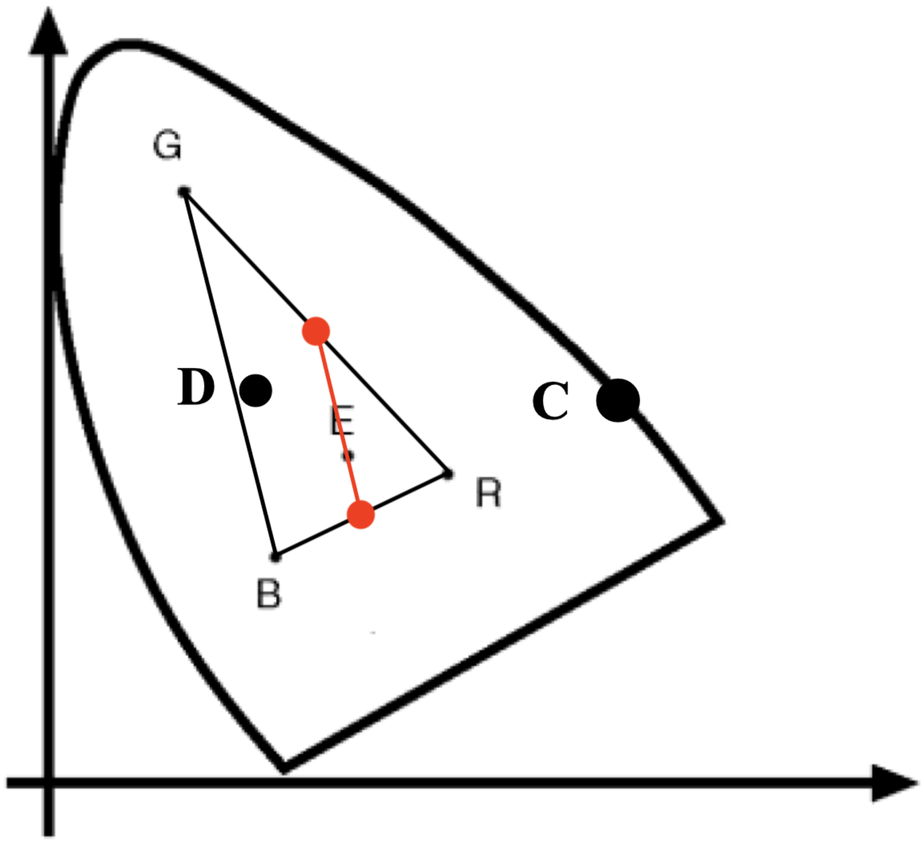
(4) Since G and B are also marked on the side image along with R, I'm assuming statements such as $R = 0.5$ mean colors that are mixed from R, G, and B and contain 50% R.



(5)

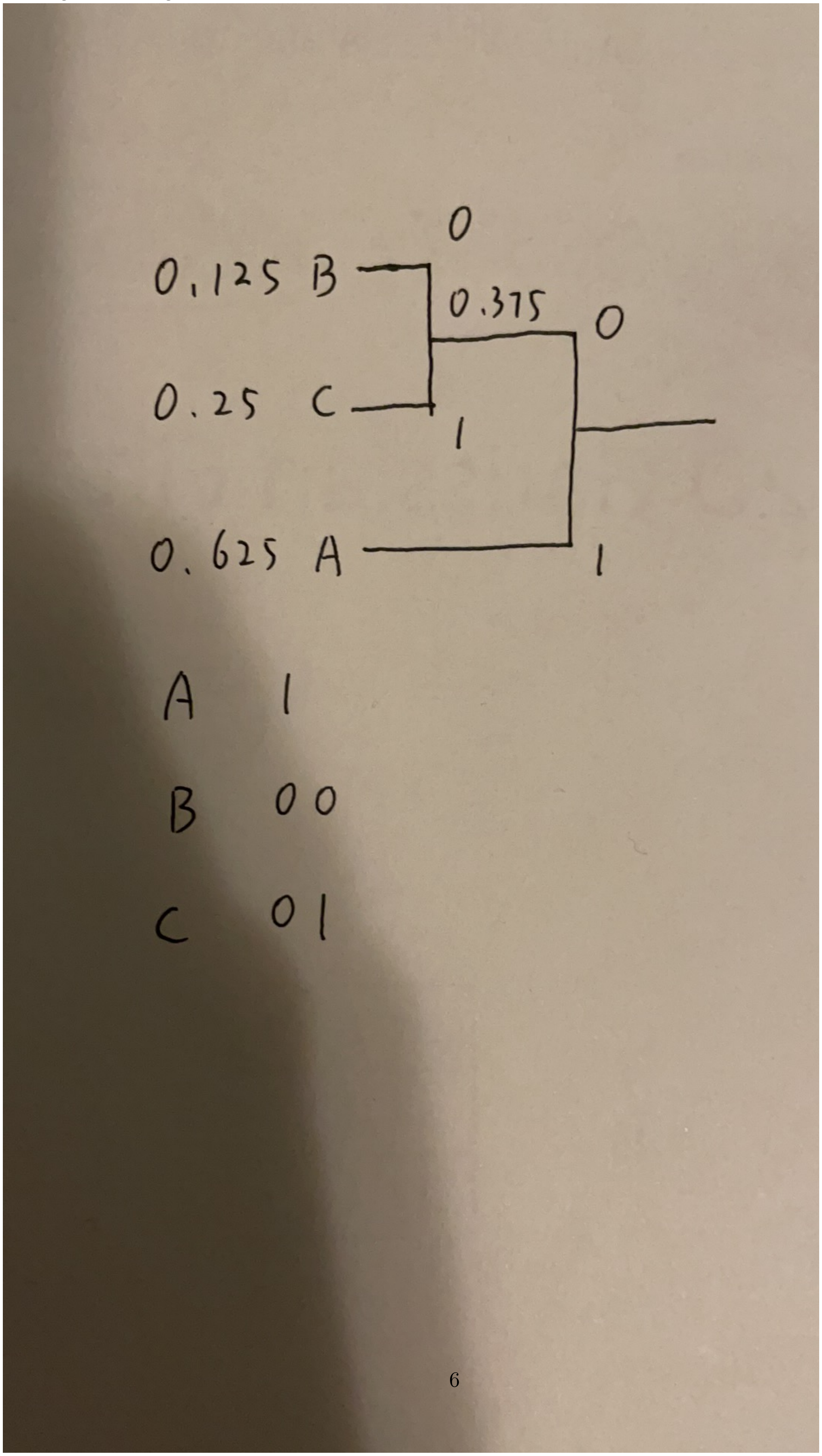


(6)


G
D
E
C
R
B

Question 2: Generic Compression

(1) Average code length is $0.625 + 0.25 * 2 + 0.125 * 2 = 1.375$



(2)

	code1	code2	code3	code4
A	0	0	1	1
B	11	10	01	00
C	10	11	00	01

(3) No; the entropy for this data is $-(0.625 * log_2(0.625) + 0.25 * log_2(0.25) + 0.125 * log_2(0.125)) \approx 1.299 < 1.375$, so not optimal. One improvement can be to encode 2 at a time. This produces the encoding shown below: The resulting code length, averaged to each signal, is $1.32 < 1.375$.

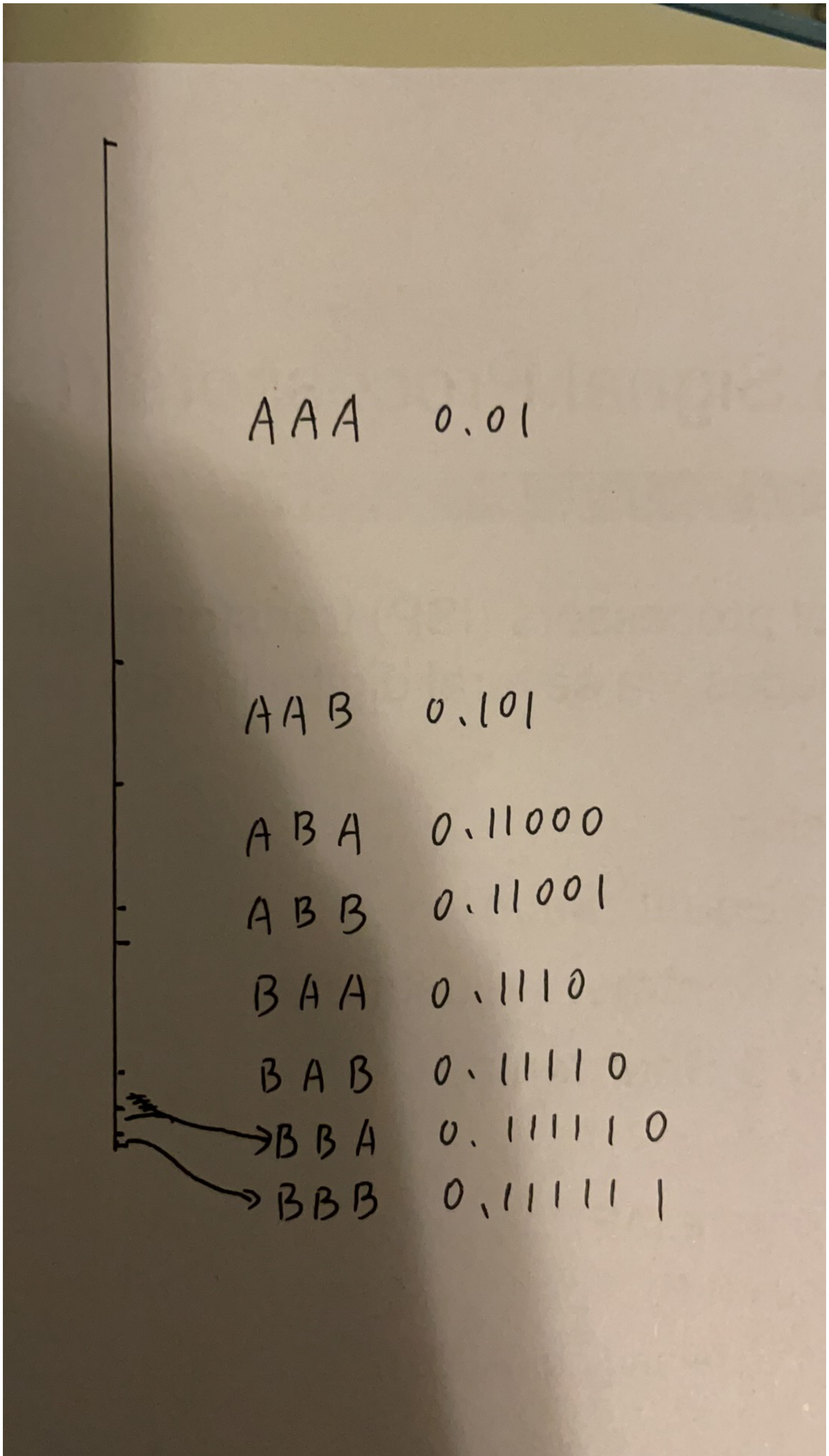
	prob	code
AA	0.390625	0
AB	0.078125	1010
AC	0.15625	110
BA	0.078125	1011
BB	0.015625	100110
BC	0.03125	100111
CA	0.15625	111
CB	0.03125	10010
CC	0.0625	1000

Question 3: Arithmetic Compression

(1)

AAA	0.512
AAB	0.128
ABA	0.128
ABB	0.032
BAA	0.128
BAB	0.032
BBA	0.032
BBB	0.008

(2)

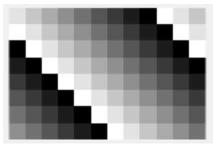

AAA 0.01
AAB 0.101
ABA 0.11000
ABB 0.11001
BAA 0.1110
BAB 0.11110
BBA 0.111110
BBB 0.111111

(3) $\frac{0.512*2+0.128*(3+5+4)+0.032*(5+5+6)+0.008*6}{3} = 1.04$

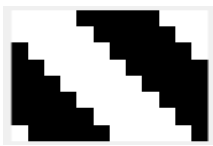
Question 4: Image Dithering

I have the MATLAB script in a file called $simple_dithering.m$.

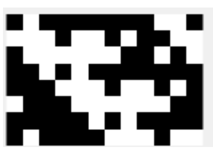
(1)



(2)



(3)



(4)

Implementation Notes

For Successive Bit Approximation, I assume "most significant bit" means the highest non-sign non-zero bit. So I first construct a height-by-width-by-3 matrix containing the location of this most significant bit for each pixel and each channel (using log_2), and then I do a while loop where for each iteration, I bit shift each pixel by its stored most significant bit location and decrement the location (since we are progressively using more bits). The while loop runs until all locations become 0, i.e. no more information to add. Some pixels have higher most significant bits, so those pixels take longer to settle while pixels with lower most significant bits stabilize in fewer iterations.