T-61.5100 Digital image processing, Exercise 8/07

1.

a) IGS (Improved Gray-Scale Quantization) coding adds pseudorandom noise to an image. The noise is generated from the low-order bits of pixel's gray-level values. This added noise reduces false contouring in decoded images.

orig.	*32*	*16*	*8*	*4*	*2*	*1*
12	0	0	1	1	0	0
12	0	0	1	1	0	0
13	0	0	1	1	0	1
13	**0**	**0**	**1**	**1**	**0**	**1**
10	0	0	1	0	1	0
13	0	0	1	1	0	1
57	1	1	1	0	0	1
54	1	1	0	1	1	0

32	*16*	*8*	*4*	*2*	*1*	*IGS*
0	0	1	1	0	0	8
0	1	0	0	0	0	16
0	0	1	**1**	**0**	**1**	8
0	1	0	0	1	0	16
0	0	1	1	0	0	8
0	1	0	0	0	1	16
1	1	1	0	0	1	56
1	1	0	1	1	1	48

On the left are the original gray-levels and their binary representations. On the right are the quantized gray-levels and their binary representations. We start by copying the first row of bits from left to right. Then we add the 3 lowest-order bits to the bits on the second row on the left. This results in the bits on the second row on the right. This procedure is then continued, e.g. the bolded bits on the third row are added to the bolded bits on the fourth row, and thus we get the bits on the fourth row on the right.
If the higher-order bits on the left are full ('111'), then we copy the whole row to the right like in the seventh row. After we are done, the three higher-order bits on the right are the IGS code.

b) The probabilities for the gray levels are:

$$p(13) = 3/8 \qquad p(12) = 2/8 \qquad p(10) = p(54) = p(57) = 1/8$$

The most probable gray-levels are coded with less bits than the rare ones. The code must also be unambiguous. We can choose e.g.

Gray-level	p(Gray-level)	Code
13	3/8	1
12	2/8	01
10	1/8	001
54	1/8	0000
57	1/8	0001

The average code lengths are obtained with Eq. 8.1-4:
a) 3
b) $1 \cdot 3/8 + 2 \cdot 2/8 + 3 \cdot 1/8 + 4 \cdot 1/8 + 4 \cdot 1/8 = 2.25$

The plain binary code: 6.

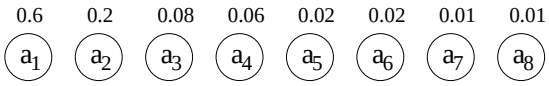
The redundancy between pixels could be reduced with e.g. run-length coding. We could also use differences between neighboring pixels. The difference row would now be

$$\{12,\ 0,\ 1,\ 0,\ -3,\ 3,\ 44,\ -3\}.$$

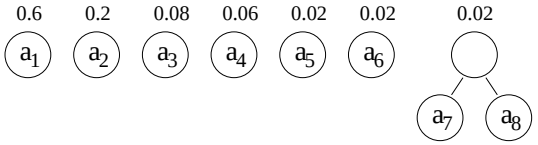
Differences are usually gathered around the origin. However, when we are dealing with such a short and varying sample, we do not gain any advantage when compared to a normal coding.

2.

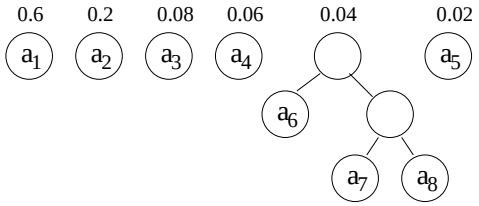
a) The Huffman code is constructed as follows. In the beginning, each symbol is its own tree whose weight is the probability of the symbol. Then with each step we combine two 'lightest' trees (if there are several possibilities, we can choose any of them). The weight of the new tree is the sum of the weights of the combined trees. This procedure is repeated until we end up with only one tree.



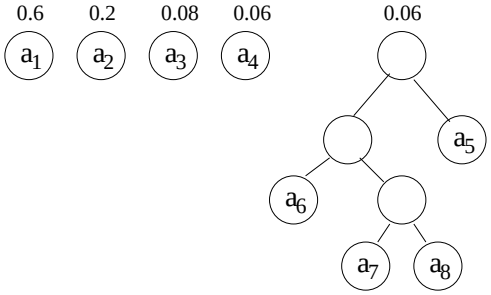
Beginning



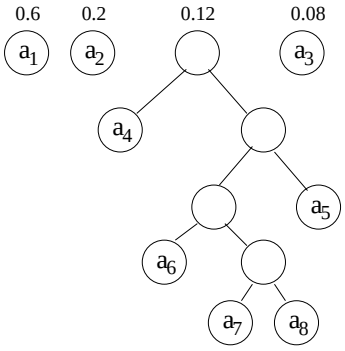
Step 1.



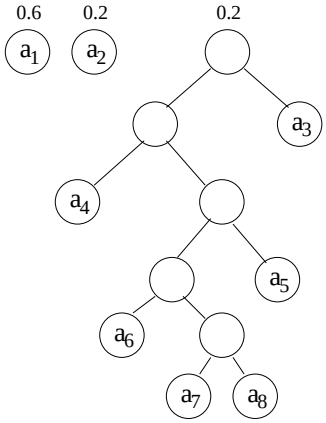
Step 2.



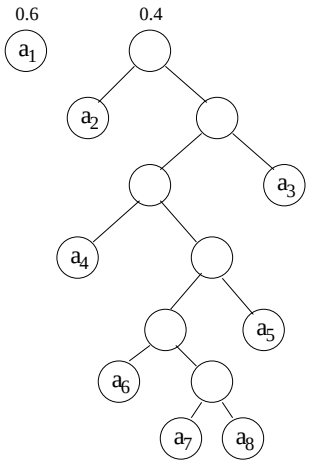
Step 3.



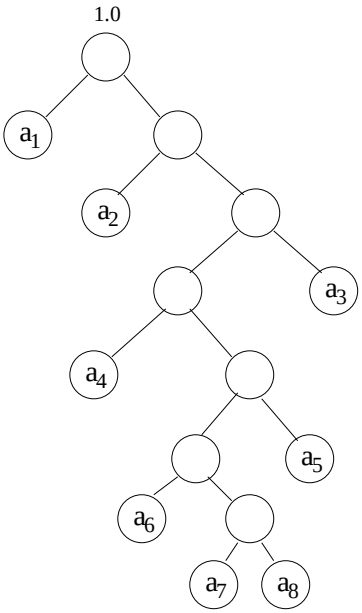
Step 4.



Step 5.



Step 6.



The resulting tree.

Now the binary code of each symbol is the path from the root node to the symbol node. The path to the left is marked with '0 and to the right with '1'. So we obtain:

$$\begin{aligned} a_1 &= 0 \\ a_2 &= 10 \\ a_3 &= 111 \\ a_4 &= 1100 \\ a_5 &= 11011 \\ a_6 &= 110100 \\ a_7 &= 1101010 \\ a_8 &= 1101011 \end{aligned}$$

b) In B_2 code we have an additional bit C. It tells us whether we continue with the current symbol or with the next symbol.

$$\begin{aligned} a_1 &= C00 \\ a_2 &= C01 \\ a_3 &= C10 \\ a_4 &= C11 \\ a_5 &= C00\ C00 \\ a_6 &= C00\ C01 \\ a_7 &= C00\ C10 \\ a_8 &= C00\ C11 \end{aligned}$$

c) In S_2 code we reserve a block of two bits to tell us if we will continue with the current symbol.

$$\begin{aligned} a_1 &= 00 \\ a_2 &= 01 \\ a_3 &= 10 \\ a_4 &= 11\ 00 \\ a_5 &= 11\ 01 \\ a_6 &= 11\ 10 \\ a_7 &= 11\ 11\ 00 \\ a_8 &= 11\ 11\ 01 \end{aligned}$$

The entropy is

$$H = -\sum_{i=1}^{8} P(a_i) \log_2 P(a_i) = -[0.6 \log_2 0.6 + \cdots + 0.01 \log_2 0.01] = 1.80 \text{ bits/symbol}$$

and the average word lengths are

a) $0.6 \cdot 1 + 0.2 \cdot 2 + 0.08 \cdot 3 + 0.06 \cdot 4 + 0.02 \cdot 5 + 0.02 \cdot 6 + 0.01 \cdot 7 + 0.01 \cdot 7 = 1.84$ bits/symbol.

b) $3 \cdot (0.6 + 0.2 + 0.08 + 0.06) + 6 \cdot (0.02 + 0.02 + 0.01 + 0.01) = 3.18$ bits/symbol.

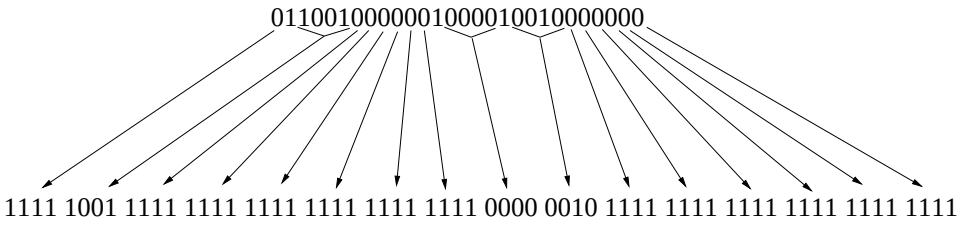
c) $2 \cdot (0.6 + 0.2 + 0.08) + 4 \cdot (0.06 + 0.02 + 0.02) + 6 \cdot (0.01 + 0.01) = 2.28$ bits/symbol.

Huffman code uses less bits (on the average) for each symbol so it can be considered optimal in that sense. As we can see from the results, the word length of Huffman code is very close to entropy. B_2 code is not well suited for this exercise since it uses at least three bits for each symbol. Since we are now having only 8 symbols, they can always be coded with three bits (the plain binary code).

3.

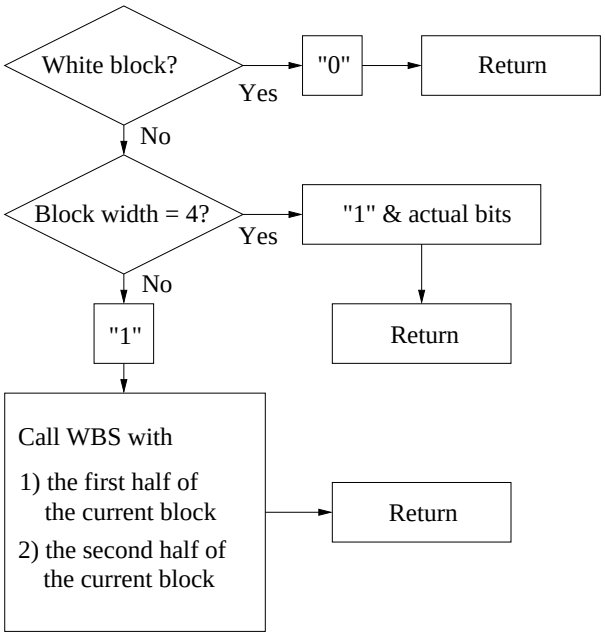
In WBS (White block skipping) code the most common white pixel lines are marked with a short code, e.g. '0'. For other lines we need to have a prefix, e.g. '1'.

a)

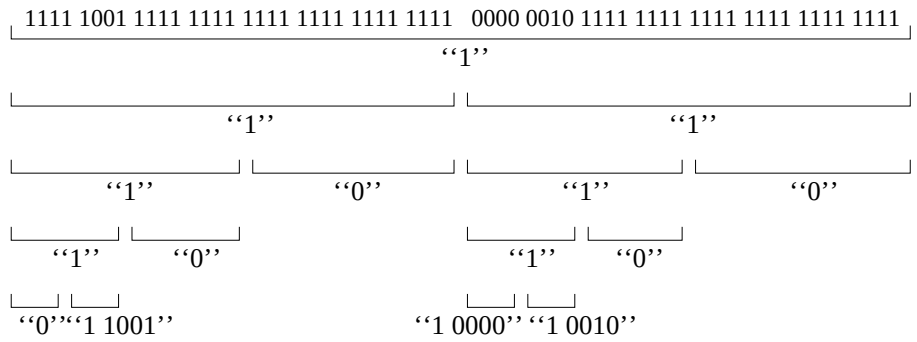


'1' stands for a white pixel in the decoded string and '0' for a black pixel.

b) A recursive program WBS, whose argument is the block that has to be encoded:



c) The given string is encoded:



The encoded result is now:

$$11110110010011110000 1001000$$

The code length is 27 bits. The original code had 28 bits, so the new code is a little shorter.

4.

The input to the LZW decoding algorithm is

$$39\ 39\ 126\ 126\ 256\ 258\ 260\ 259\ 257\ 126$$

The starting dictionary, to be consistent with the coding itself, contains 512 locations - with the first 256 corresponding to gray level values 0 through 255. The decoding algorithm begins by getting the first encoded value, outputting the corresponding value from the dictionary, and setting the “recognized sequence” to the first value. For each additional encoded value, we (1) output the dictionary entry for the pixel value(s), (2) add a new dictionary entry whose content is the “recognized sequence” plus the first element of the encoded value being processed, and (3) set the “recognized sequence” to the encoded value being processed. For the encoded output in Example 8.12, the sequence of operations is:

Recognized	Encoded Value	Pixels	Dict. Address	Dict. Entry
	39	39		
39	39	39	256	39-39
39	126	126	257	39-126
126	126	126	258	126-126
126	256	39-39	259	126-39
256	258	126-126	260	39-39-126
258	260	39-39-126	261	126-126-39
260	259	126-39	262	39-39-126-126
259	257	39-126	263	126-39-39
257	126	126	264	39-126-126

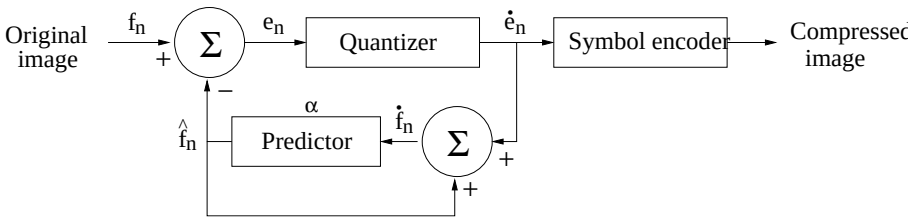
Note, for example, in row 5 of the table that the new dictionary entry for location 259 is 126-39, the concatenation of the currently recognized sequence, 126, and the first element of the encoded value being processed - the 39 from the 39-39 entry in dictionary location 256. The output is then read from the third column of the table to yield

$$\begin{matrix} 39 & 39 & 126 & 126 \\ 39 & 39 & 126 & 126 \\ 39 & 39 & 126 & 126 \\ 39 & 39 & 126 & 126 \end{matrix}$$

where it is assumed that the decoder knows or is given the size of the image that was received. Note that the dictionary is generated as the decoding is carried out.

5.

The encoder:



The predictor:

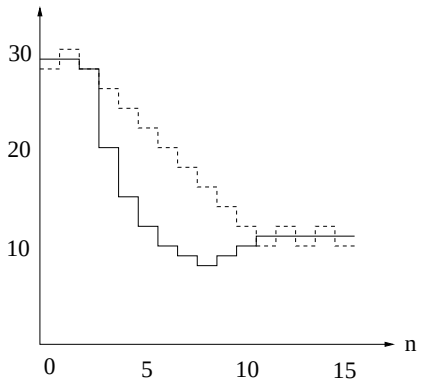
$$\hat{f}_n = \alpha \dot{f}_{n-1}, \quad \alpha = 1$$

The quantizer:

$$\dot{e}_n = \begin{cases} +2, & \text{when } e_n \geq 0 \\ -2, & \text{when } e_n < 0 \end{cases}$$

The initial values are $f_0 = \dot{f}_0 = 30$.

f_n	29	29	28	20	15	12	10	9	8	9	10	11	11	11	11	11
$\hat{f}_n$	30	28	30	28	26	24	22	20	18	16	14	12	10	12	10	12
e	-1	$+1$	-2	-8	-11	-12	-12	-11	-10	-7	-4	-1	$+1$	-1	$+1$	-1
$\dot{e}$	-2	$+2$	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	$+2$	-2	$+2$	-2
$\dot{f}$	28	30	28	26	24	22	20	18	16	14	12	10	12	10	12	10



It can be seen from the above image that DM cannot follow rapid changes and it will produce noise even in flat areas. The encoding will thus loose information.
In non-lossy or information preserving coding the error must be encoded exactly. If the predictor is given as

$$\hat{f}_n = \alpha f_{n-1} = f_{n-1}$$

then the error is

$$e_n = f_n - \hat{f}_n = f_n - f_{n-1}.$$

The largest change will determine the code length. In the example the largest change is $20-28 = -8$. So we need 4 bits to encode the change if it is assumed that the change can also happen in another direction to $+7$. In DM only one bit is required.