2.16 The QM Coder

JPEG (Section 4.8) is an important image compression method. It uses arithmetic coding, but not in the way described in Section 2.14. The arithmetic coder of JPEG is called the QM-coder and is described in this section. It is designed for simplicity and speed, so it is limited to input symbols that are single bits and it uses an approximation instead of a multiplication. It also uses fixed-precision integer arithmetic, so it has to resort to *renormalization* of the probability interval from time to time, in order for the approximation to remain close to the true multiplication. For more information on this method, see [IBM 88], [Pennebaker and Mitchell 88a], and [Pennebaker and Mitchell 88b].

A slight confusion arises because the arithmetic coder of JPEG 2000 (Section 5.19) and JBIG2 (Section 4.12) is called the MQ-coder and is not the same as the QM-coder (the author is indebted to Christopher M. Brislawn for pointing this out).

The QM-coder is limited to input symbols that are single bits. However, it is possible to convert an arbitrary set of symbols to a stream of bits. Perhaps the simplest way to do this is to calculate a set of Huffman codes for the symbols, using their probabilities. This converts each symbol to a binary string, so the input stream can be encoded by the QM-coder. After the compressed stream is decoded by the QM-decoder, an extra step is needed, to convert the resulting binary strings back to the original symbols.

The main idea behind the QM-coder is to classify each input symbol (which is a single bit) as either the more probable symbol (MPS) or the less probable symbol (LPS). Before the next bit is input, the QM-encoder uses a statistical model to determine whether a 0 or a 1 is more probable at that point. It then inputs the next bit and classifies it according to its actual value. If the model predicts, for example, that a 0 is more probable, and the next bit turns out to be a 1, the encoder classifies it as an LPS. It is important to understand that the only information encoded in the compressed stream is whether the next bit is MPS or LPS. When the stream is decoded, all that the decoder knows is whether the bit that has just been decoded is an MPS or an LPS. The decoder has to use the same statistical model to determine the current relation between MPS/LPS and 0/1. This relation changes, of course, from bit to bit, since the model is updated identically (in lockstep) by the encoder and decoder each time a bit is input by the former or decoded by the latter.

The statistical model also computes a probability Qe for the LPS, so the probability of the MPS is $(1-Qe)$. Since Qe is the probability of the *less probable* symbol, it is in the range $[0, 0.5]$. The encoder divides the probability interval A into two subintervals according to Qe and places the LPS subinterval (whose size is $A\times Qe$) above the MPS subinterval [whose size is $A(1-Qe)$], as shown in Figure 2.60b. Notice that the two subintervals in the figure are closed at the bottom and open at the top. This should be compared with the way a conventional arithmetic encoder divides the same interval (Figure 2.60a, where the numbers are taken from Table 2.47).

In conventional arithmetic coding, the interval is narrowed all the time, and the final output is any number inside the final subinterval. In the QM-coder, for simplicity, each step adds the bottom of the selected subinterval to the output-so-far. We denote the output string by C. If the current bit read from the input is the MPS, the bottom of the MPS subinterval (i.e., the number 0) is added to C. If the current bit is the LPS,

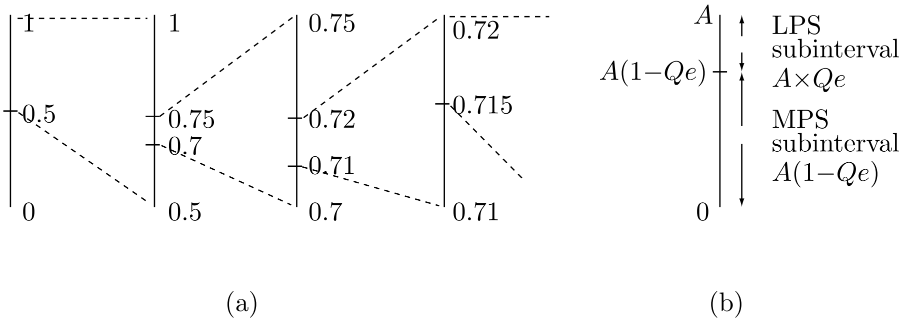

Figure 2.60: Division of the Probability Interval.

the bottom of the LPS subinterval [i.e., the number $A(1-Qe)$] is added to C. After C is updated in this way, the current probability interval A is shrunk to the size of the selected subinterval. The probability interval is always in the range $[0, A)$, and A gets smaller at each step. This is the main principle of the QM-encoder, and it is expressed by the rules

$$\begin{aligned} \text{After MPS:}\quad & C \text{ is unchanged},\quad A \leftarrow A(1-Qe),\\ \text{After LPS:}\quad & C \leftarrow C + A(1-Qe),\quad A \leftarrow A\times Qe. \end{aligned} \tag{2.7}$$

These rules set C to point to the bottom of the MPS or the LPS subinterval, depending on the classification of the current input bit. They also set A to the new size of the subinterval.

Table 2.61 lists the values of A and C when four symbols, each a single bit, are encoded. We assume that they alternate between an LPS and an MPS and that $Qe = 0.5$ for all four steps (normally, of course, the statistical model yields different values of Qe all the time). It is easy to see how the probability interval A shrinks from 1 to 0.0625, and how the output C grows from 0 to 0.625. Table 2.63 is similar, but uses $Qe = 0.1$ for all four steps. Again A shrinks, to 0.0081, and C grows, to 0.981. Figures 2.62 and 2.64 illustrate graphically the division of the probability interval A into an LPS and an MPS.

◇ **Exercise 2.17:** Repeat these calculations for the case where all four symbols are LPS and $Qe = 0.5$, then for the case where they are MPS and $Qe = 0.1$.

The principle of the QM-encoder is simple and easy to understand, but it involves two problems. The first is the fact that the interval A, which starts at 1, shrinks all the time and requires high precision to distinguish it from zero. The solution to this problem is to maintain A as an integer and double it every time it gets too small. This is called *renormalization*. It is fast, since it is done by a logical left shift; no multiplication is needed. Each time A is doubled, C is also doubled. The second problem is the multiplication $A\times Qe$ used in subdividing the probability interval A. A fast compression method should avoid multiplications and divisions and should try to replace them with additions, subtractions, and shifts. It turns out that the second problem is also solved by renormalization. The idea is to keep the value of A close to 1, so that Qe will not be very different from the product $A\times Qe$. The multiplication is *approximated* by Qe.

Symbol	C	A
Initially	0	1
s1 (LPS)	$0+1(1-0.5)=0.5$	$1\times0.5=0.5$
s2 (MPS)	unchanged	$0.5\times(1-0.5)=0.25$
s3 (LPS)	$0.5+0.25(1-0.5)=0.625$	$0.25\times0.5=0.125$
s4 (MPS)	unchanged	$0.125\times(1-0.5)=0.0625$

Table 2.61: Encoding Four Symbols With $Qe=0.5$.

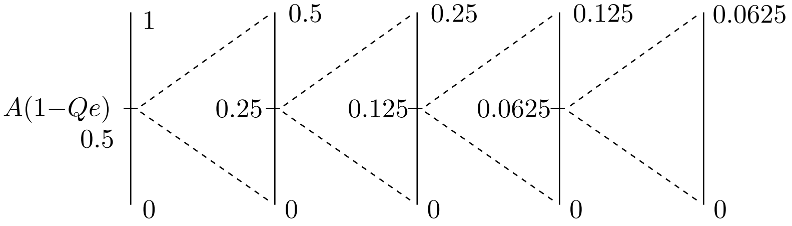

Figure 2.62: Division of the Probability Interval.

Symbol	C	A
Initially	0	1
s1 (LPS)	$0+1(1-0.1)=0.9$	$1\times0.1=0.1$
s2 (MPS)	unchanged	$0.1\times(1-0.1)=0.09$
s3 (LPS)	$0.9+0.09(1-0.1)=0.981$	$0.09\times0.1=0.009$
s4 (MPS)	unchanged	$0.009\times(1-0.1)=0.0081$

Table 2.63: Encoding Four Symbols With $Qe=0.1$.

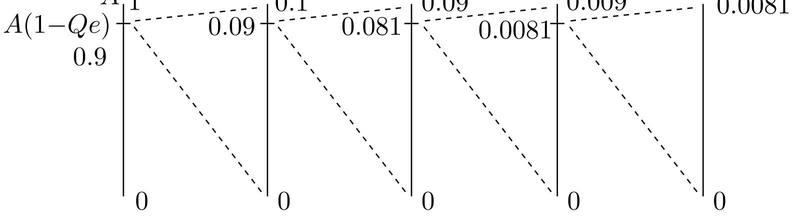

Figure 2.64: Division of the Probability Interval.

How can we use renormalization to keep A close to 1? The first idea that comes to mind is to double A when it gets just a little below 1, say to 0.9. The problem is that doubling 0.9 yields 1.8, closer to 2 than to 1. If we let A get below 0.5 before doubling it, the result will be less than 1. It does not take long to realize that 0.75 is a good minimum value for renormalization. If A reaches this value at a certain step, it is doubled, to 1.5. If it reaches a smaller value, such as 0.6 or 0.55, it ends up even closer to 1 when doubled.

If A reaches a value less than 0.5 at a certain step, it has to be renormalized by doubling it several times, each time also doubling C. An example is the second row of Table 2.63, where A shrinks from 1 to 0.1 in one step, because of a very small probability Qe. In this case, A has to be doubled three times, from 0.1 to 0.2, to 0.4, to 0.8, in order to bring it into the desired range $[0.75, 1.5)$. We conclude that A can go down to 0 (or very close to 0) and can be at most 1.5 (actually, less than 1.5, since our intervals are always open at the high end).

If the current input bit is an LPS, A is shrunk to Qe, which is always 0.5 or less, so A always has to be renormalized in such a case.

Approximating the multiplication $A\times Qe$ by Qe changes the main rules of the QM-encoder to

$$\begin{aligned}
&\text{After MPS:}\quad C \text{ is unchanged},\quad A \leftarrow A(1-Qe) \approx A - Qe,\\
&\text{After LPS:}\quad C \leftarrow C + A(1-Qe) \approx C + A - Qe,\quad A \leftarrow A\times Qe \approx Qe.
\end{aligned}$$

In order to include renormalization in these rules, we have to choose an integer representation for A where real values in the range $[0, 1.5)$ are represented as integers. Since many current computers have 16-bit words, it makes sense to choose a representation where 0 is represented by a word of 16 zero bits and 1.5 is represented by the smallest 17-bit number, which is

$$2^{16} = 65536_{10} = 10000_{16} = 1\underbrace{0\ldots0}_{16}{}_2.$$

This way we can represent 65536 real values in the range $[0, 1.5)$ as 16-bit integers, where the largest 16-bit integer, 65535, represents a real value slightly less than 1.5. Here are a few important examples of such values:

$$\begin{gathered}
0.75 = 1.5/2 = 2^{15} = 32768_{10} = 8000_{16},\quad 1 = 0.75(4/3) = 43690_{10} = AAAA_{16},\\
0.5 = 43690/2 = 21845_{10} = 5555_{16},\quad 0.25 = 21845/2 = 10923_{10} = 2AAB_{16}.
\end{gathered}$$

(The optimal value of 1 in this representation is $AAAA_{16}$, but the way we associate the real values of A with the 16-bit integers is somewhat arbitrary. The important thing about this representation is to achieve accurate interval subdivision, and the subdivision is done by either $A \leftarrow A - Qe$ or $A \leftarrow Qe$. The accuracy of the subdivision depends, therefore, on the relative values of A and Qe, and it has been found experimentally that the average value of A is $B55A_{16}$, so this value, instead of $AAAA_{16}$, is associated in the JPEG QM-coder with $A = 1$. The difference between the two values $AAAA$ and $B55A$ is $AB0_{16} = 2736_{10}$. The JBIG QM-coder uses a slightly different value for 1.)

Renormalization can now be included in the main rules of the QM-encoder, which become

$$\begin{aligned}\text{After MPS: } & C \text{ is unchanged}, \quad A \leftarrow A - Qe,\\ & \text{if } A < 8000_{16} \text{ renormalize } A \text{ and } C.\\ \text{After LPS: } & C \leftarrow C + A - Qe, \quad A \leftarrow Qe,\\ & \text{renormalize } A \text{ and } C.\end{aligned} \tag{2.8}$$

Tables 2.65 and 2.66 list the results of applying these rules to the examples shown in Tables 2.61 and 2.63, respectively.

Symbol	C	A	Renor. A	Renor. C
Initially	0	1		
s1 (LPS)	$0+1-0.5=0.5$	0.5	1	1
s2 (MPS)	unchanged	$1-0.5=0.5$	1	2
s3 (LPS)	$2+1-0.5=2.5$	0.5	1	5
s4 (MPS)	unchanged	$1-0.5=0.5$	1	10

Table 2.65: Renormalization Added to Table 2.61.

Symbol	C	A	Renor. A	Renor. C
Initially	0	1		
s1 (LPS)	$0+1-0.1=0.9$	0.1	0.8	$0.9{\cdot}2^3=7.2$
s2 (MPS)	unchanged 7.2	$0.8-0.1=0.7$	1.4	$7.2{\cdot}2=14.4$
s3 (LPS)	$14.4+1.4-0.1=15.7$	0.1	0.8	$15.7{\cdot}2^3=125.6$
s4 (MPS)	unchanged	$0.8-0.1=0.7$	1.4	$125.6{\cdot}2=251.2$

Table 2.66: Renormalization Added to Table 2.63.

◇ **Exercise 2.18:** Repeat these calculations with renormalization for the case where all four symbols are LPS and $Qe = 0.5$. Following this, repeat the calculations for the case where they are all MPS and $Qe = 0.1$. (Compare this exercise with Exercise 2.17.)

The next point that has to be considered in the design of the QM-encoder is the problem of *interval inversion.* This is the case where the size of the subinterval allocated to the MPS becomes smaller than the LPS subinterval. This problem may occur when Qe is close to 0.5, and is a result of the approximation to the multiplication. It is illustrated in Table 2.67, where four MPS symbols are encoded with $Qe = 0.45$. In the third row of the table the interval A is doubled from 0.65 to 1.3. In the fourth row it is reduced to 0.85. This value is greater than 0.75, so no renormalization takes place, yet the subinterval allocated to the MPS becomes $A - Qe = 0.85 - 0.45 = 0.40$, which is smaller than the LPS subinterval, which is $Qe = 0.45$. Clearly, the problem occurs when $Qe > A/2$, a relation that can also be expressed as $Qe > A - Qe$.

The solution is to interchange the two subintervals whenever the LPS subinterval becomes greater than the MPS subinterval. This is called *conditional exchange.* The condition for interval inversion is $Qe > A - Qe$, but since $Qe \le 0.5$, we get $A - Qe <$

Symbol	C	A	Renor. A	Renor. C
Initially	0	1		
s1 (MPS)	0	$1-0.45=0.55$	1.1	0
s2 (MPS)	0	$1.1-0.45=0.65$	1.3	0
s3 (MPS)	0	$1.3-0.45=0.85$		
s4 (MPS)	0	$0.85-0.45=0.40$	0.8	0

Table 2.67: Illustrating Interval Inversion.

$Qe \le 0.5$, and it becomes obvious that both Qe and $A-Qe$ (i.e., both the LPS and MPS subintervals) are less than 0.75, so renormalization must take place. This is why the test for conditional exchange is performed only *after* the encoder has decided that renormalization is needed. The new rules for the QM-encoder are shown in Figure 2.68.

```
After MPS:
   $C$ is unchanged
   $A \leftarrow A-Qe$;          % The MPS subinterval
   if $A < 8000_{16}$ then % if renormalization needed
    if $A < Qe$ then      % if inversion needed
    $C \leftarrow C+A$;           % point to bottom of LPS
    $A \leftarrow Qe$             % Set $A$ to LPS subinterval
    endif;
   renormalize $A$ and $C$;
   endif;

After LPS:
   $A \leftarrow A-Qe$;          % The MPS subinterval
   if $A \ge Qe$ then % if interval sizes not inverted
    $C \leftarrow C+A$;           % point to bottom of LPS
    $A \leftarrow Qe$             % Set $A$ to LPS subinterval
   endif;
   renormalize $A$ and $C$;
```

Figure 2.68: QM-Encoder Rules With Interval Inversion.

The QM-Decoder: The QM-decoder is the reverse of the QM-encoder. For simplicity we ignore renormalization and conditional exchange, and we assume that the QM-encoder operates by the rules of Equation (2.7). Reversing the way C is updated in those rules yields the rules for the QM-decoder (the interval A is updated in the same way):

$$\begin{aligned}&\text{After MPS:}\quad C \text{ is unchanged},\quad A \leftarrow A(1-Qe),\\ &\text{After LPS:}\quad C \leftarrow C - A(1-Qe),\quad A \leftarrow A\times Qe.\end{aligned} \tag{2.9}$$

These rules are demonstrated using the data of Table 2.61. The four decoding steps are as follows:

Step 1: $C = 0.625$, $A = 1$, the dividing line is $A(1 - Qe) = 1(1 - 0.5) = 0.5$, so the LPS and MPS subintervals are $[0, 0.5)$ and $[0.5, 1)$. Since C points to the upper subinterval, an LPS is decoded. The new C is $0.625 - 1(1-0.5) = 0.125$ and the new A is $1 \times 0.5 = 0.5$.
Step 2: $C = 0.125$, $A = 0.5$, the dividing line is $A(1 - Qe) = 0.5(1 - 0.5) = 0.25$, so the LPS and MPS subintervals are $[0, 0.25)$ and $[0.25, 0.5)$, and an MPS is decoded. C is unchanged and the new A is $0.5(1 - 0.5) = 0.25$.
Step 3: $C = 0.125$, $A = 0.25$, the dividing line is $A(1 - Qe) = 0.25(1 - 0.5) = 0.125$, so the LPS and MPS subintervals are $[0, 0.125)$ and $[0.125, 0.25)$, and an LPS is decoded. The new C is $0.125 - 0.25(1 - 0.5) = 0$ and the new A is $0.25 \times 0.5 = 0.125$.
Step 4: $C = 0$, $A = 0.125$, the dividing line is $A(1 - Qe) = 0.125(1 - 0.5) = 0.0625$, so the LPS and MPS subintervals are $[0, 0.0625)$ and $[0.0625, 0.125)$, and an MPS is decoded. C is unchanged and the new A is $0.125(1 - 0.5) = 0.0625$.

◇ **Exercise 2.19:** Use the rules of Equation (2.9) to decode the four symbols encoded in Table 2.63.

Probability Estimation: The QM-encoder uses a novel, interesting, and little-understood method for estimating the probability Qe of the LPS. The first method that comes to mind in trying to estimate the probability of the next input bit is to initialize Qe to 0.5 and update it by counting the numbers of zeros and ones that have been input so far. If, for example, 1000 bits have been input so far, and 700 of them were zeros, then 0 is the current MPS, with probability 0.7, and the probability of the LPS is $Qe = 0.3$. Notice that Qe should be updated *before* the next input bit is read and encoded, since otherwise the decoder would not be able to mirror this operation (the decoder does not know what the next bit is). This method produces good results, but is slow, since Qe should be updated often (ideally, for each input bit), and the calculation involves a division (dividing 700/1000 in our example).

The method used by the QM-encoder is based on a table of preset Qe values. Qe is initialized to 0.5 and is modified when renormalization takes place, not for every input bit. Table 2.69 illustrates the process. The Qe index is initialized to zero, so the first value of Qe is $0AC1_{16}$ or very close to 0.5. After the first MPS renormalization, the Qe index is incremented by 1, as indicated by column "Incr MPS." A Qe index of 1 implies a Qe value of $0A81_{16}$ or 0.49237, slightly smaller than the original, reflecting the fact that the renormalization occurred because of an MPS. If, for example, the current Qe index is 26, and the next renormalization is LPS, the index is decremented by 3, as indicated by column "Decr LPS," reducing Qe to 0.00421. The method is not applied very often, and it involves only table lookups and incrementing or decrementing the Qe index: fast, simple operations.

The column labeled "MPS exch" in Table 2.69 contains the information for the conditional exchange of the MPS and LPS definitions at $Qe = 0.5$. The zero value at the bottom of column "Incr MPS" should also be noted. If the Qe index is 29 and an MPS renormalization occurs, this zero causes the index to stay at 29 (corresponding to the smallest Qe value).

Table 2.69 is used here for illustrative purposes only. The JPEG QM-encoder uses Table 2.70, which has the same format but is harder to understand, since its Qe values are not listed in sorted order. This table was prepared using probability estimation concepts based on Bayesian statistics (see appendix in book's web site).

Qe index	Hex Qe	Dec Qe	Decr LPS	Incr MPS	MPS exch	Qe index	Hex Qe	Dec Qe	Decr LPS	Incr MPS	MPS exch
0	0AC1	0.50409	0	1	1	15	0181	0.07050	2	1	0
1	0A81	0.49237	1	1	0	16	0121	0.05295	2	1	0
2	0A01	0.46893	1	1	0	17	00E1	0.04120	2	1	0
3	0901	0.42206	1	1	0	18	00A1	0.02948	2	1	0
4	0701	0.32831	1	1	0	19	0071	0.02069	2	1	0
5	0681	0.30487	1	1	0	20	0059	0.01630	2	1	0
6	0601	0.28143	1	1	0	21	0053	0.01520	2	1	0
7	0501	0.23456	2	1	0	22	0027	0.00714	2	1	0
8	0481	0.21112	2	1	0	23	0017	0.00421	2	1	0
9	0441	0.19940	2	1	0	24	0013	0.00348	3	1	0
10	0381	0.16425	2	1	0	25	000B	0.00201	2	1	0
11	0301	0.14081	2	1	0	26	0007	0.00128	3	1	0
12	02C1	0.12909	2	1	0	27	0005	0.00092	2	1	0
13	0281	0.11737	2	1	0	28	0003	0.00055	3	1	0
14	0241	0.10565	2	1	0	29	0001	0.00018	2	0	0

Table 2.69: Probability Estimation Table (Illustrative).

We now justify this probability estimation method with an approximate calculation that suggests that the Qe values obtained by this method will adapt to and closely approach the correct LPS probability of the binary input stream. The method updates Qe each time a renormalization occurs, and we know, from Equation (2.8), that this happens every time an LPS is input, but not for all MPS values. We therefore imagine an ideal balanced input stream where for each LPS bit there is a sequence of consecutive MPS bits. We denote the true (but unknown) LPS probability by q, and we try to show that the Qe values produced by the method for this ideal case are close to q.

Equation (2.8) lists the main rules of the QM-encoder, and shows how the probability interval A is decremented by Qe each time an MPS is input and encoded. Imagine a renormalization that brings A to a value A_1 (between 1 and 1.5), followed by a sequence of N consecutive MPS bits that reduce A in steps of Qe from A_1 to a value A_2 that requires another renormalization (i.e., A_2 is less than 0.75). It is clear that

$$N = \left\lfloor \frac{\Delta A}{Qe} \right\rfloor,$$

where $\Delta A = A_1 - A_2$. Since q is the true probability of an LPS, the probability of having N MPS bits in a row is $P = (1-q)^N$. This implies $\ln P = N \ln(1-q)$, which, for a small q, can be approximated by

$$\ln P \approx N(-q) = -\frac{\Delta A}{Qe} q, \text{ or } P \approx \exp\left(-\frac{\Delta A}{Qe} q\right). \tag{2.10}$$

Since we are dealing with an ideal balanced input stream, we are interested in the value $P = 0.5$, because it implies equal numbers of LPS and MPS renormalizations. From

Qe	Hex	Next-Index		MPS
index	*Qe*	LPS	MPS	exch
0	5A1D	1	1	1
1	2586	14	2	0
2	1114	16	3	0
3	080B	18	4	0
4	03D8	20	5	0
5	01DA	23	6	0
6	00E5	25	7	0
7	006F	28	8	0
8	0036	30	9	0
9	001A	33	10	0
10	000D	35	11	0
11	0006	9	12	0
12	0003	10	13	0
13	0001	12	13	0
14	5A7F	15	15	1
15	3F25	36	16	0
16	2CF2	38	17	0
17	207C	39	18	0
18	17B9	40	19	0
19	1182	42	20	0
20	0CEF	43	21	0
21	09A1	45	22	0
22	072F	46	23	0
23	055C	48	24	0
24	0406	49	25	0
25	0303	51	26	0
26	0240	52	27	0
27	01B1	54	28	0
28	0144	56	29	0
29	00F5	57	30	0
30	00B7	59	31	0
31	008A	60	32	0
32	0068	62	33	0
33	004E	63	34	0
34	003B	32	35	0
35	002C	33	9	0
36	5AE1	37	37	1
37	484C	64	38	0
38	3A0D	65	39	0
39	2EF1	67	40	0
40	261F	68	41	0
41	1F33	69	42	0
42	19A8	70	43	0
43	1518	72	44	0
44	1177	73	45	0
45	0E74	74	46	0
46	0BFB	75	47	0
47	09F8	77	48	0
48	0861	78	49	0
49	0706	79	50	0
50	05CD	48	51	0
51	04DE	50	52	0
52	040F	50	53	0
53	0363	51	54	0
54	02D4	52	55	0
55	025C	53	56	0
56	01F8	54	57	0
57	01A4	55	58	0
58	0160	56	59	0
59	0125	57	60	0
60	00F6	58	61	0
61	00CB	59	62	0
62	00AB	61	63	0
63	008F	61	32	0
64	5B12	65	65	1
65	4D04	80	66	0
66	412C	81	67	0
67	37D8	82	68	0
68	2FE8	83	69	0
69	293C	84	70	0
70	2379	86	71	0
71	1EDF	87	72	0
72	1AA9	87	73	0
73	174E	72	74	0
74	1424	72	75	0
75	119C	74	76	0
76	0F6B	74	77	0
77	0D51	75	78	0
78	0BB6	77	79	0
79	0A40	77	48	0
80	5832	80	81	1
81	4D1C	88	82	0
82	438E	89	83	0
83	3BDD	90	84	0
84	34EE	91	85	0
85	2EAE	92	86	0
86	299A	93	87	0
87	2516	86	71	0
88	5570	88	89	1
89	4CA9	95	90	0
90	44D9	96	91	0
91	3E22	97	92	0
92	3824	99	93	0
93	32B4	99	94	0
94	2E17	93	86	0
95	56A8	95	96	1
96	4F46	101	97	0
97	47E5	102	98	0
98	41CF	103	99	0
99	3C3D	104	100	0
100	375E	99	93	0
101	5231	105	102	0
102	4C0F	106	103	0
103	4639	107	104	0
104	415E	103	99	0
105	5627	105	106	1
106	50E7	108	107	0
107	4B85	109	103	0
108	5597	110	109	0
109	504F	111	107	0
110	5A10	110	111	1
111	5522	112	109	0
112	59EB	112	111	1

Table 2.70: The QM-Encoder Probability Estimation Table.

$P = 0.5$ we get $\ln P = -\ln 2$, which, when combined with Equation (2.10), yields

$$Qe = \frac{\Delta A}{\ln 2} q.$$

This is fortuitous because $\ln 2 \approx 0.693$ and ΔA is typically a little less than 0.75. We can say that for our ideal balanced input stream, $Qe \approx q$, providing one justification for our estimation method. Another justification is provided by the way P depends on Qe [shown in Equation (2.10)]. If Qe gets larger than q, P also gets large, and the table tends to move to smaller Qe values. In the opposite case, the table tends to select larger Qe values.

2.17 Text Compression

Before delving into the details of the next method, here is a general discussion of text compression. Most text compression methods are either statistical or dictionary based. The latter class breaks the text into fragments that are saved in a data structure called a dictionary. When a fragment of new text is found to be identical to one of the dictionary entries, a pointer to that entry is written on the compressed stream, to become the compression of the new fragment. The former class, on the other hand, consists of methods that develop statistical *models* of the text.

A common statistical method consists of a modeling stage followed by a coding stage. The model assigns probabilities to the input symbols, and the coding stage then actually codes the symbols based on those probabilities. The model can be static or dynamic (adaptive). Most models are based on one of the following two approaches.

Frequency: The model assigns probabilities to the text symbols based on their frequencies of occurrence, such that commonly occurring symbols are assigned short codes. A static model uses fixed probabilities, whereas a dynamic model modifies the probabilities "on the fly" while text is being input and compressed.

Context: The model considers the context of a symbol when assigning it a probability. Since the decoder does not have access to future text, both encoder and decoder must limit the context to past text, i.e., to symbols that have already been input and processed. In practice, the context of a symbol is the N symbols preceding it. We thus say that a context-based text compression method uses the context of a symbol to **predict** it (i.e., to assign it a probability). Technically, such a method is said to use an "order-N" Markov model. The PPM method, Section 2.18, is an excellent example of a context-based compression method, although the concept of context can also be used to compress images.

Some modern context-based text compression methods perform a transformation on the input data and then apply a statistical model to assign probabilities to the transformed symbols. Good examples of such methods are the Burrows-Wheeler method, Section 8.1, also known as the Burrows-Wheeler transform, or *block sorting*; the technique of symbol ranking, Section 8.2; and the ACB method, Section 8.3, which uses an associative dictionary.