The arithmetic coders used in JPEG, JPEG 2000 and JBIG are called QM-coder\(^1\). It handles only binary strings or input and it is designed for simplicity and speed. It uses approximation for multiplication operation, fixed-precision integer arithmetic with renormalization of the probability interval from time to time.

The main idea of the QM-coder is to classify the input bit as **More Probable Symbol (MPS)** and **Less Probable Symbol (LPS)**. Before the next bit is input, the QM-coder uses a statistical model (using a context, typically a two-dimensional context of black and white pixel in an image) to predict which one of the bits (0 or 1) will be the MPS. If the predicted MPS bit does not match with the actual bit, then the QM-coder will classify this as LPS; otherwise, it will continue to be classified as MPS. The output of the coder is simply a compressed version of a stream of MPS or LPS, which are assigned probability values dynamically. The decoder has only the knowledge of whether the next predicted bit is MPS or LPS. It uses the same statistical model as that of the encoder to obtain the actual values of the bit. Recall the range update equations we used for arithmetic coding: Let \( L \) and \( H \) denote the current ‘low’ and ‘high’, respectively, and current ‘range’ \( A = H - L \). Let the current incoming symbol be \( a_i \), its probability be \( p(a_i) \) and its cumulative low and high probability be \( P(a_{i-1}) \) and \( P(a_i) \), respectively, then the new low and new high become

\[
L: = L + (H - L) * P(a_{i-1})
\]

\[
H: = L + (H - L) * P(a_i)
\]

Then the new range becomes (by subtracting new \( L \) from new \( H \))

\[
A: = A * [P(a_i) - P(a_{i-1})]
\]

Let us assign a probability \( Q \) to LPS and assign the lower interval to MPS with

\(^1\) There is an error in the book [Data Compression: The Complete Reference, 2nd Edition]: Page 121, Table 2.64: The renormalized values of C are wrong. The correct table is:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>C</th>
<th>A</th>
<th>Renor. A</th>
<th>Renor. C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially 0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s1 (LPS)</td>
<td>0+1=0.9</td>
<td>0.1</td>
<td>0.8</td>
<td>0.9*2^3= 7.2</td>
</tr>
<tr>
<td>s2 (MPS)</td>
<td>unchanged 7.2</td>
<td>0.8-0.1=0.7</td>
<td>1.4</td>
<td>7.2*2=14.4</td>
</tr>
<tr>
<td>s3 (LPS)</td>
<td>14.4+1.4=15.7</td>
<td>0.1</td>
<td>0.8</td>
<td>15.7*2^3=125.6</td>
</tr>
<tr>
<td>s4 (MPS)</td>
<td>unchanged 125.6</td>
<td>0.8-0.1=0.7</td>
<td>1.4</td>
<td>125.6*2=251.2</td>
</tr>
</tbody>
</table>

Table 2.64 Renormalization Added to Table 2.61.

--------From http://www.davidsalomon.name/DC2advertis/DC2errata.html
The value of Q is calculated dynamically using a statistical model. The output is simply a fractional value C indicating the state of the encoder, more specifically, the lower bound of the new sub-interval. If the next symbol is classified as MPS, the lower bound of the new interval remains unchanged, so C is unchanged. However, if the next symbol is LPS, the lower bound of the new sub-interval will be leveraged by the value of A(1-Q), so C is updated as C+A(1-Q).

The encoder executes the following code:

\[
\text{When MPS is encoded} \\
\text{begin} \\
\quad C \text{ is unchanged} ; \\
\quad A := A (1 – Q) \\
\text{End}
\]

\[
\text{When LPS is encoded} \\
\text{begin} \\
\quad C := C+A (1 – Q) ; \\
\quad A := AQ \\
\text{End}
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPS</td>
<td>Q</td>
<td>[1-Q,1]</td>
</tr>
<tr>
<td>MPS</td>
<td>1-Q</td>
<td>[0,1-Q]</td>
</tr>
</tbody>
</table>

Example:

Assume \( Q= 0.5 \). Input: (LPS, MPS, LPS, MPS). Initially, \( C=0 \) and \( A=1 \).
Step 1: $\text{LPS}: C = 0 + 1(1 - 0.5) = 0.5$

$A = 1 \cdot 0.5 = 0.5$

Step 2: $\text{MPS}: C = 0.5$

$A = 0.5(1 - 0.5) = 0.25$

Step 3: $\text{LPS}: C = 0.5 + 0.25(1 - 0.5) = 0.625$

$A = 0.25 \cdot 0.5 = 0.125$

Step 4: $\text{MPS}: C = 0.625$

$A = 0.125(1 - 0.5) = 0.0625$

Note, the range continues to shrink and the value of $C$ gets bigger with longer input sequence.

The QM-decoder is just the reverse of the QM-encoder. We begin with initial value of $A = 1$. The dividing line is $A(1 - Q) = 1(1 - 0.5) = 0.5$. So, initially the $(\text{MPS}, \text{LPS})$ subintervals are $[0, 0.5)$ and $[0.5, 1)$, respectively.

Step 1: $C = 0.625$ falls in upper subinterval. So, an $\text{LPS}$ is decoded. The new $C = 0.625 - 1(1 - 0.5) = 0.125$ and new $A = 1 \cdot 0.5 = 0.5$.

Step 2: $C = 0.125$, $A = 0.5$. The dividing line is $0.5(1 - 0.5) = 0.25$. The $(\text{MPS}, \text{LPS})$ subintervals are $[0, 0.25)$ and $[0.25, 0.5)$, respectively. So, a $\text{MPS}$ is decoded. The value of $C$ remains unchanged and new $A = 0.5(1 - 0.05) = 0.25$.

Step 3: $C = 0.125$, $A = 0.25$. The dividing line is $0.25(1 - 0.5) = 0.125$. The $(\text{MPS}, \text{LPS})$ subintervals are $[0, 0.125)$ and $[0.125, 0.25)$, respectively. So, a $\text{LPS}$ is decoded. The new $C = 0.125 - 0.25(1 - 0.5) = 0$ and new $A = 0.25 \cdot 0.5 = 0.125$.

Step 4: $C = 0$, $A = 0.125$. The dividing line is $0.125(1 - 0.5) = 0.0625$. The $(\text{MPS}, \text{LPS})$ subintervals are $[0, 0.0625)$ and $[0.0625, 0.125)$, respectively. So, a $\text{MPS}$ is decoded. The value of $C$ remains unchanged and new $A = 0.125(1 - 0.05) = 0.0625$.

The decoding algorithm can be written as

```
Input $C$, $A = 1$ and $Q$;
Begin
    $X := A \cdot (1 - Q)$;
    MPS_interval = $[0, X)$;
    LPS_interval = $[X, A)$;
    If $C$ is in MPS_interval then
        { Output MPS symbol;
          $A := A \cdot (1 - Q)$
        }
    else
        { Output LPS symbol;
          $C := C - A \cdot (1 - Q)$;
          $A := A \cdot Q$
        }
    endif;
```

If we take $Q_e = 0.1$, with the same input, the final value of $C$ will be 0.981 and that of $A$ 0.0081. If $A$ gets too small, the precision of the machine will not be able to handle the fraction. The proposed remedy is to renormalize by doubling $A$ and $C$ (in effect, left shifting the binary integer by one bit representing the multiplication by 2). So, we need some threshold $t$ such that when $A$ is less than $t$, then $A$ should be doubled. Furthermore, both $A$ and $2^*A$ should be close to 1.0. The mathematically representation for this requirement is:

Find $t > 0.5$ such that we have $\min(\max(1-t, 2t -1))$

If $t$ has the value of 0.9, then $\max(1-t, 2t -1) = 0.8$; if $t$ has the value of 0.6, then $\max(1-t, 2t -1) = 0.4$. The solution to this problem is $t = 0.75$, and $\max(1-t, 2t -1) = 0.25$.

The other problem is to compute the new value of the range $A$ which needs multiplication. The JBIG committee recommended that the multiplication could be avoided by approximating the value of $A$ to be close to 1, that is, $A \cdot (1 - Q_e)$ is approximated by $(A - Q_e)$ and $A Q_e$ by $Q_e$. With the optimal threshold $t = 0.75$, the value of $A$ is always maintained between [0.75, 1.5).

In order not to violate the assumption that $A$ is close to 1, whenever $A$ goes below 0.75 the QM coder goes through a series of rescaling until the value of $A$ goes higher than 0.75. The rescaling operation is simply doubling which corresponds to a left shift if $A$ is represented in binary. The same rescaling must be applied to $C$ by doubling its value. Rescaling happens every time LPS occurs. For $A$, the rescaling occurs only if its value dips below 0.75. The modified encoder algorithm is:

**When MPS is encoded**

begin
    $C$ is unchanged;
    $A := A - Q_e$;
    If ($A < 0.75$) then
        Renormalize $A$ and $C$;
    Endif;
End

**When LPS is encoded**

begin
    $C := C + (A - Q_e)$;
    $A := Q_e$;
    Renormalize $A$ and $C$;
End
In the aforementioned pseudo-code, when LPS is encoded, A is updated as Qe, which is in practice always less than 0.5. So the condition of the re-normalization \( A < 0.75 \) holds. Therefore re-normalization is always conducted when we have a LPS symbol.

One problem introduced by approximated re-normalization is that the subinterval allocated to the MPS may be smaller than the sub-interval allocated to the LPS. This is because we use \( A - Qe \) to approximate \( A = A(1-Qe) \). For example, suppose we have \( Qe = 0.45 \), and we have four MPS in a row. Initially, we have \( C = 0, A = 1 \). After first MPS, A becomes 0.55, and re-normalized to 1.1, while C remains 0. After second MPS, A becomes 0.65, and re-normalized to 1.3, while C remains 0. After third MPS, A becomes 0.85, and no re-normalization is performed. When we encode the fourth MPS, the sub-interval allocated to the MPS becomes \( A - Qe = 0.40 \), while the sub-interval allocated to LPS is 0.45. Now, the sub-interval allocated to MPS is smaller than the LPS sub-interval. It can be seen that the condition for this situation is that \( Qe > A / 2 \), or \( Qe > A - Qe \). The solution is to perform conditional exchange: interchanging the two intervals under the aforementioned condition. Because conditional exchange is caused by the approximated re-normalization, the test for conditional exchange is only performed when re-normalization is needed.

The pseudo-code for the approximated re-normalization with conditional exchange is:

**When MPS is encoded**

begin
- C is unchanged;
- \( A; = A - Qe; \)
- If \( A < 0.75 \) then
  - If \( A < Qe \) then
    - \( C := C + A \)
    - \( A = Qe \)
  - Endif
- Renormalize A and C;
- Endif
End

**When LPS is encoded**

begin
- If \( A - Qe >= Qe \) then
  - \( C := C + A - Q \)
  - \( A = Qe \)
- Else
  - \( A = A - Qe \)
- Endif
- Renormalize A and C;
End