Information Theory. 2nd Chapter Slides

Boris Kudryashov

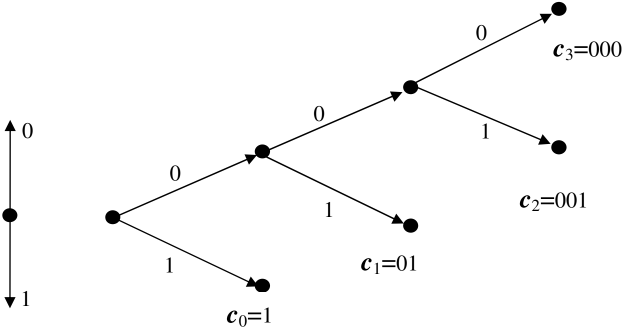
ITMO University

January 9, 2017

Agenda

1. Uneven letter-by-letter coding
2. Kraft inequality
3. Letter-by-letter coding theorems
4. Huffman code
5. Huffman code redundancy
6. Shannon code
7. Gilbert-Moore code
8. Stationary source coding

Example: $X = \{0, 1, 2, 3\}$

- $C_1 = \{00, 01, 10, 11\}$;
- $C_2 = \{1, 01, 001, 000\}$;
- $C_3 = \{1, 10, 100, 000\}$;
- $C_4 = \{0, 1, 10, 01\}$;
- 

- Consider $X = \{1, ..., M\}$, $\{p_1, ..., p_M\}$.
  $C = \{\boldsymbol{c}_1, ..., \boldsymbol{c}_M\}$, codewords $l_1, \ldots, l_M$.
- Average codeword length

$$\bar{l} = \mathbf{M}[l_i] = \sum_{i=1}^{M} p_i l_i$$

Theorem

Kraft inequality Prefix code of size M with codewords of length $l_1, \ldots, l_M$ exists iff Kraft inequality holds:

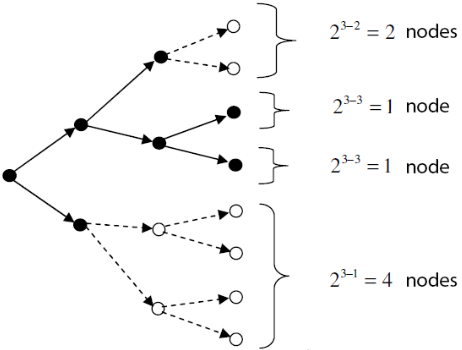
$$\sum_{i=1}^{M} 2^{-l_i} \leq 1. \tag{1}$$

Proof of theorem

- Consider L, such that $L \geq \max_i l_i$.

$$\sum_{i=1}^{M} 2^{L-l_i} \leq 2^L.$$

- Consider:



Proof of theorem

- $$2^{l_2} - 2^{l_2 - l_1} \geq 1$$
- $$2^{l_3} - 2^{l_3 - l_2} - 2^{l_3 - l_1} \geq 1$$
- $$2^{l_M} - 2^{l_M - l_{M-1}} - 2^{l_M - l_{M-2}} - \ldots - 2^{l_M - l_1}$$

- $l_1 = 1,\ l_2 = 2,\ l_3 = l_4 = 3.$

$l_1 = 1$. $n = 2^{l_1} = 2$. From n nodes choose one node for word of length $l_1 = 1$.

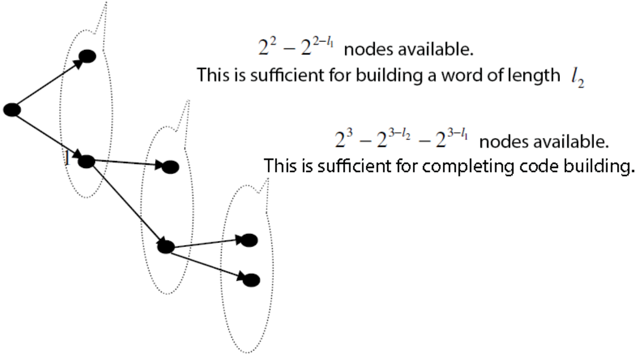


Figure: Binary code tree construction.

Theorem

For any uniquely decoded binary code of size M with codeword length $l_1, \ldots, l_M$ holds

$$\sum_{i=1}^{M} 2^{-l_i} \leq 1. \tag{2}$$

Proof of Theorem

- Consider $N \in \mathbb{N}$

$$\left(\sum_{i=1}^{M} 2^{-l_i}\right)^N = \underbrace{\left(\sum_{i_1=1}^{M} 2^{-l_{i_1}}\right) \dots \left(\sum_{i_N=1}^{M} 2^{-l_{i_N}}\right)}_{N \text{ factors}} =$$

$$= \sum_{i_1=1}^{M} \dots \sum_{i_N=1}^{M} 2^{-(l_{i_1}+\dots+l_{i_N})}.$$

- $$\left(\sum_{i=1}^{M} 2^{-l_i}\right)^N = \sum_{L=1}^{Nl_M} A_L 2^{-L}.$$

- $$\left(\sum_{i=1}^{M} 2^{-l_i}\right)^N \le \sum_{L=1}^{Nl_M} 2^L 2^{-L} = Nl_M.$$
- N-th root:

$$\sum_{i=1}^{M} 2^{-l_i} \le (Nl_M)^{1/N} = 2^{\frac{\log(Nl_M)}{M}}.$$

Theorem

Achievability Theorem. For ensemble $X = \{x, p(x)\}$ with entropy H there exists letter-by-letter uneven prefix code with average codeword length $\bar{l} < H + 1$.

Achievability Theorem Proof

- Consider $X = \{1, \ldots, M\}$, $p_1, \ldots, p_M$.
- $x_m < - > l_m = \lceil -\log p_m \rceil$, $m = 1, \ldots, M$.
- Codeword length satisfies Kraft inequality:

$$\sum_{m=1}^{M} 2^{-l_m} = \sum_{m=1}^{M} 2^{-\lceil -\log p_m \rceil} \leq$$

$$\leq \sum_{m=1}^{M} 2^{\log p_m} = \sum_{m=1}^{M} p_m = 1.$$

Achievability Theorem Proof

- Average codeword length:

$$
\begin{aligned}
\bar{l} &= \sum_{m=1}^{M} p_m l_m = \\
&= \sum_{m=1}^{M} p_m \lceil -\log p_m \rceil < \\
&< \sum_{m=1}^{M} p_m (-\log p_m + 1) = \\
&= H + \sum_{m=1}^{M} p_m = \\
&= H + 1,
\end{aligned}
$$

Theorem

Inverse Theorem. For any uniquely decoded code of discrete source $X = \{x, p(x)\}$ *with entropy* H*, average codeword length* $\bar{l}$ *satisfies:*

$$\bar{l} \geq H. \tag{3}$$

Proof Inverse Theorem

- Let $l(x)$ be the codeword length for message x.

$$H - \bar{l} = -\sum_{x\in X} p(x)\log p(x) - \sum_{x\in X} p(x)l(x) = \sum_{x\in X} p(x)\log \frac{2^{-l(x)}}{p(x)}.$$

- Use Kraft inequality:

$$\log x \le (x-1)\log e,$$

- We get

$$\begin{aligned} H - \bar{l} &\le \log e \sum_{x\in X} p(x)\left(\frac{2^{-l(x)}}{p(x)} - 1\right) = \\ &= \log e\left(\sum_{x\in X} 2^{-l(x)} - \sum_{x\in X} p(x)\right) \le \\ &\le \log e\left(1 - \sum_{x\in X} p(x)\right) = 0. \qquad (4) \end{aligned}$$

Properties of Huffman code

1. If $p_i < p_j$, the $l_i \geq l_j$.
2. At least two codewords have the same langth $l_M = \max_m l_m$.
3. There are two codewords of length $l_M = \max_m l_m$ which differ only in the last character.
4. If code C' for X' is optimal, then, code C is optimal for X.

Proof of property 4

$$l_m = \begin{cases} l'_m \text{ при } m \le M-2, \\ l'_{M-1} + 1 \text{ при } m = M-1, M. \end{cases}$$

$$\begin{aligned}
\bar{l} &= \sum_{m=1}^{M} p_m l_m = \\
&= \sum_{m=1}^{M-2} p_m l_m + p_{M-1} l_{M-1} + p_M l_M = \\
&= \sum_{m=1}^{M-2} p_m l_m + (p_{M-1} + p_M)(l'_{M-1} + 1) = \\
&= \sum_{m=1}^{M-2} p'_m l'_m + p'_{M-1} l'_{M-1} + p_{M-1} + p_M = \\
&= \sum_{m=1}^{M-1} p'_m l'_m + p_{M-1} + p_M = \bar{l}' + p_{M-1} + p_M.
\end{aligned}$$

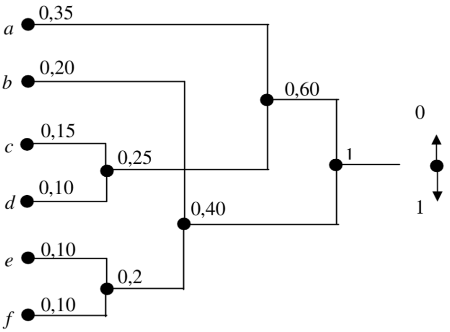
Input: Alphabet size M, probabilities of characters
Output: Bitary tree of Huffman code

`Init:` number of unprocessed nodes $M_0 = M$ **while** $M_0 > 1$ **do**

Find two unprocessed nodes with min probabilities. Exclude these nodes from the list of unprocessed. Introduce new node, attribute to him the total probability of two excluded nodes. Bind new node with edges to two excluded node. $M_0 \leftarrow M_0 - 1$.

end

Figure: Building Huffman code tree



Letter	Codeword
a	00
b	10
c	010
d	011
e	110
f	111

Figure: Huffman code example

Theorem

Let p_1 be maximal probability of message. Then redundancy of Huffman code for this ensemble satisfies:

$$r \leq \begin{cases} p_1 + \sigma, & \text{for } p_1 < 1/2, \\ 2 - \eta(p_1) - p_1 & \text{for } p_1 \geq 1/2, \end{cases} \tag{5}$$

where $\eta(x) = -x \log x - (1 - x) \log(1 - x)$ is binary ensemble entropy, and

$$\sigma = 1 - \log e - \log \log e \approx 0,08607. \tag{6}$$

Huffman code redundancy

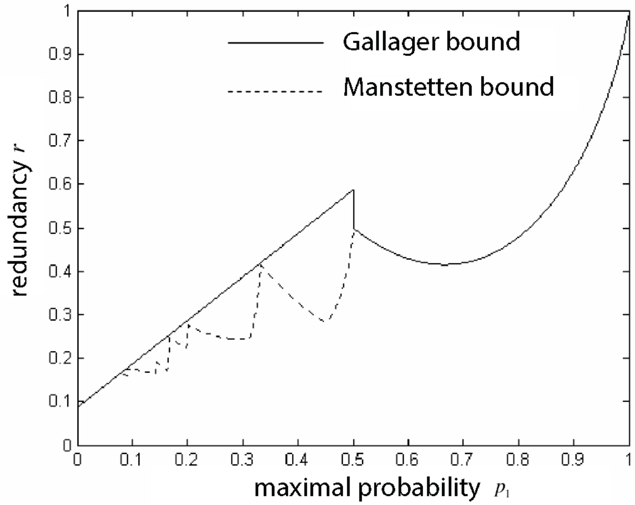


Figure: Huffman code redundancy

Input: Alphabet size M, character probabilities $p_i, i = 1, ..., M$
Output: List of Shannon codewords

```
Sorting:
```
for $i = 1$ **to** M **do**
  $j(i) \leftarrow$ index of i-th descending character probability
end

```
Cumulative probabilities:
```
$q_{j(1)} = 0$; **for** $i = 2$ **to** M **do**
  $q_{j(i)} = q_{j(i-1)} + p_{j(i-1)}$;
end

```
Codewords:
```
for $i = 1$ **to** M **do**
  $\boldsymbol{c}_i \leftarrow$ first $\lceil -\log p_i \rceil$ bits after comma in a binary number q_i.
end

Figure: Shannon code construction algorithm

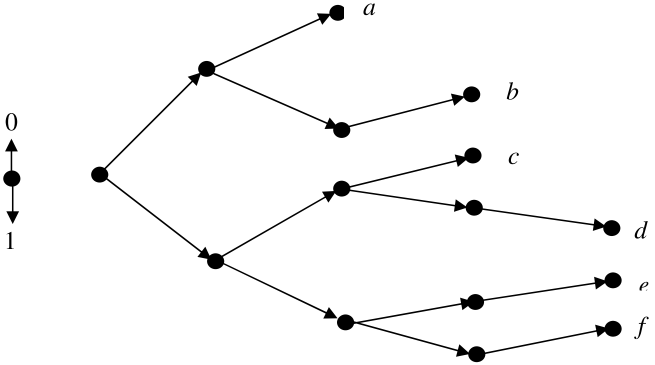


Figure: Shannon codetree for $X = \{a, b, c, d, e, f\}$, $\{0,35, 0,2, 0,15, 0,1, 0,1, 0,1\}$

Table: Shannon codetree building for
$X = \{a, b, c, d, e, f\}$, $\{0,35, 0,2, 0,15, 0,1, 0,1, 0,1\}$

x	p_m	q_m	l_m	Binary notation q_m	Codeword $\boldsymbol{c}_m$
a	0,35	0,00	2	0,00...	00
b	0,20	0,35	3	0,0101...	010
c	0,15	0,55	3	0,10001...	100
d	0,10	0,70	4	0,10110...	1011
e	0,10	0,80	4	0,11001...	1100
f	0,10	0,90	4	0,11100...	1110

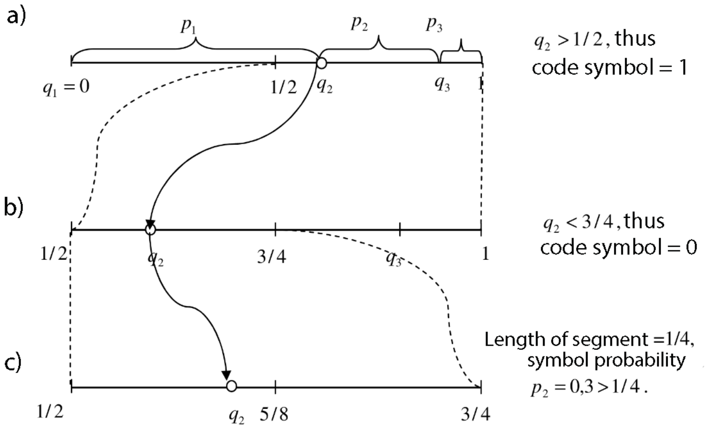


Figure: Graphical interpretation of Shannon code

Input: Alphabet size M, character probabilities $p_i, i = 1, ..., M$

Output: List of Gilbert-Moore code

`Auxiliary probabilities:` $q_1 = 0$; **for** $i = 2$ **to** M **do**

$q_i = q_{i-1} + p_{i-1}$; $\sigma_i = q_i + p_i/2$;

end

`Codewords:` **for** $i = 1$ **to** M **do**

$\boldsymbol{c}_i \leftarrow$ первые $\lceil -\log p_i \rceil + 1$ bits after comma in a binary number σ_i.

end

Figure: Gilbert-Moore code constriction algorithm

Table: Gilbert-Moore code eample

x_m	p_m	q_m	σ_m	l_m	$\boldsymbol{c}_m$ [a]	$\tilde{\boldsymbol{c}}_m$ [b]
1	0,1	0,0=[0,00000...]	0,05=[0,00001...]	5	00001	0000
2	0,6	0,1=[0,00011...]	0,40=[0,01100...]	2	01	0
3	0,3	0,7=[0,10110...]	0,85=[0,11011...]	3	110	10

[a] Gilbert-Moore codewords

[b] Shannon codewords whithout character probability ordering

- Consider

$$\begin{aligned}
\sigma_j + \frac{p_j}{2} - \sigma_i &= \sum_{h=1}^{j-1} p_h - \sum_{h=1}^{i-1} p_h - \frac{p_i}{2} = \\
&= \sum_{h=i}^{j-1} p_h + \frac{p_j - p_i}{2} \geq \\
&\geq p_i + \frac{p_j - p_i}{2} = \\
&= \frac{p_j + p_i}{2} \geq \frac{\max\{p_i, p_j\}}{2}.
\end{aligned}$$

- For codeword length and its probability holds:

$$l_m = \left\lceil -\log \frac{p_m}{2} \right\rceil \geq -\log \frac{p_m}{2}.$$

- Thus,

$$\sigma_j - \sigma_i \geq \frac{\max\{p_i, p_j\}}{2} \geq 2^{-\min\{l_i, l_j\}},$$

- Average codeword length estimation

$$\bar{l} < H + 2.$$

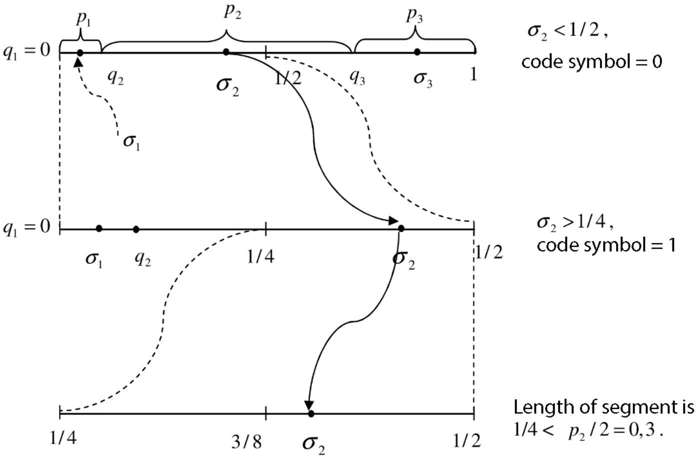


Figure: Graphical interpretation of Gilbert-Moore code.

Theorem

Inverse Theorem For DSS with entropy H for any FV-coding holds:

$$\bar{R} \geq H.$$

Proof.

- consider X^n.

$$\mathbf{M}\left[l(\boldsymbol{x})\right] \geq H(X^n) = nH_n(X) \geq nH_\infty(X) = nH.$$

- $H_n(X)$ does not increase with increasing n. Thus, $\forall n$

$$\bar{R}_n \geq H$$

-

$$\bar{R} = \inf_n R_n \geq H.$$

Theorem

Achievability theorem For DSS with entropy H and $\forall\delta > 0$ there exists FV-coding such that

$$\bar{R} \leq H + \delta.$$

Proof.

$$\mathbf{M}\left[l(\mathbf{x})\right] \leq H(X^n) + 1 = nH_n(X) + 1. \tag{7}$$

$$|H_n(X) - H| \leq \frac{\delta}{2},$$

$$H_n(X) \leq H + \frac{\delta}{2}, \quad n > n_1. \tag{8}$$

Proof.

$$\frac{1}{n} \le \frac{\delta}{2}. \qquad (9)$$

For $n \ge \max\{n_1, n_2\}$, we get

$$\begin{aligned}
\bar{R} &= \inf_m \bar{R}_m \le \\
&\le \bar{R}_n = \\
&= \frac{\mathbf{M}\left[l(x)\right]}{n} \le \\
&\le H_n(X) + \frac{1}{n} \le \\
&\le H + \frac{\delta}{2} + \frac{\delta}{2} = \\
&= H + \delta.
\end{aligned}$$

- For sequences $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ denote i to be the least index such that $x_i \neq y_i$.
- Then $\boldsymbol{y} \prec \boldsymbol{x}$, if $y_i \prec x_i$.
- Cumulative probability

$$q(\boldsymbol{x}) = \sum_{\boldsymbol{y} \prec \boldsymbol{x}} p(\boldsymbol{y}), \tag{10}$$

- For memoryless source

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i).$$

- For calculating $q(\boldsymbol{x})$.

$$
\begin{aligned}
q(\boldsymbol{x}_1^n) &= \sum_{\mathbf{y_1^n} \prec \mathbf{x_1^n}} p(\boldsymbol{y}_1^n) = \\
&= \sum_{\mathbf{y_1^{n-1}} \prec \mathbf{x_1^{n-1}}} \sum_{y_n} p(\boldsymbol{y}_1^{n-1} y_n) + \sum_{\mathbf{y_1^{n-1}} = \mathbf{x_1^{n-1}}} \sum_{y_n \prec x_n} p(\boldsymbol{y}_1^{n-1} y_n) = \\
&= \sum_{\mathbf{y_1^{n-1}} \prec \mathbf{x_1^{n-1}}} p(\boldsymbol{y}_1^{n-1}) + \sum_{\mathbf{y_1^{n-1}} = \mathbf{x_1^{n-1}}} p(\boldsymbol{y}_1^{n-1}) \sum_{y_n \prec x_n} p(y_n) = \\
&= q(\boldsymbol{x}_1^{n-1}) + p(\boldsymbol{x}_1^{n-1}) q(x_n),
\end{aligned}
$$

where $q(x_n)$ is cumulative probability of x_n.

-

$$
\begin{aligned}
q(\boldsymbol{x}_1^n) &= q(\boldsymbol{x}_1^{n-1}) + p(\boldsymbol{x}_1^{n-1}) q(x_n); && (11) \\
p(\boldsymbol{x}_1^n) &= p(\boldsymbol{x}_1^{n-1}) p(x_n). && (12)
\end{aligned}
$$

Input: Alphabet size M character probabilities $p_i, i = 1, ..., M$ length n sequence at output $(x_1, ..., x_n)$,
Output: Arithmetic codeword

`Cumulative probabilities:` $q_1 = 0$; **for** $i = 2$ **to** M **do**
  $q_i = q_{i-1} + p_{i-1}$;
end

`Coding:` **for** $i = 1$ **to** n **do**
  $F \leftarrow F + q(x_i)G$; $G \leftarrow p(x_i)G$;.
end

`Codeword creation:` $\boldsymbol{c} \leftarrow$ first $\lceil -\log G \rceil + 1$ bits after comma in a binary number $F + G/2$.

Figure: Arithmetic coding algorithm

Table: Arithmetic coding of sequence

Step i	x_i	$p(x_i)$	$q(x_i)$	F	G
0	-	-	-	0,0000	1,0000
1	b	0,6	0,1	0,1000	0,6000
2	c	0,3	0,7	0,5200	0,1800
3	b	0,6	0,1	0,5380	0,1080
4	a	0,1	0,0	0,5380	0,0108
5	b	0,6	0,1	0,5391	0,0065
6	Codeword length $\lceil -\log G + 1 \rceil = 9$ Codeword $F + G/2 = 0,5423... \rightarrow$ $\rightarrow \hat{F} = 0,541 \rightarrow 100010101$				

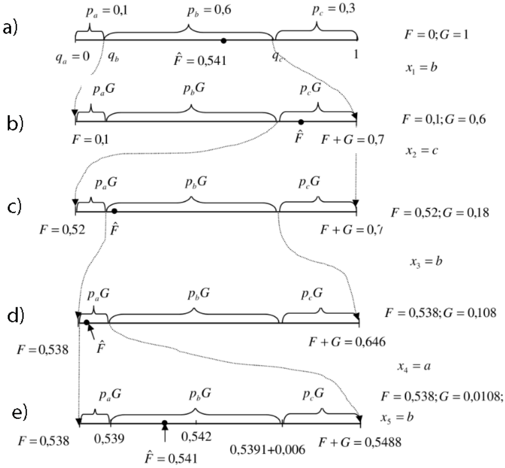


Figure: Graphical interpretation of arithmetic coding

Input: Alphabet size M cumulative probabilities $q_i, i = 1, ..., M$;
decode input $\hat{\sigma}$.
Output: Decoded character x

`Init:` $q_{M+1} = 1$; $m = 1$.;

`character search:` **while** $q_{m+1} < \hat{\sigma}$ **do**
  $m \leftarrow m + 1$.
end

`Result:` $x = x_m$

Figure: Gilbert-Moore decoding algorithm

Input: Alphabet size M;
character probabilities $\{p_1, ..., p_M\}$ cumulative probabilities $q_i, i = 1, ..., M$ decoded sequence length n, codeword as a number $\hat{F}$.
Output: Decoded character sequence $(x_1, ..., x_n)$

```
Init: q_{M+1} = 1; S = 0; G = 1.;

Decoding: ;
for i = 1 to n do
    j = 1;
    while S + q_{j+1} G < F̂ do
        j ← j + 1.
    end
    S ← S + q_j G;;
    G ← p_j G;;
    x_i = j.;
end

Result:  sequence (x_1, ..., x_n);
```

Figure: Arithmetic code decoding algorithm

Table: $X = \{a, b, c\}$. $p_a = 0,1$, $p_b = 0,6$, $p_c = 0,3$.
0100010101

Step	S	G	Hypotesis x	$q(x)$	$S + qG$	Decision x_i	$p(x)$
0	100010101→ $\hat{F}$ =0,541						
1	0,0000	1,0000	a	0,0	0,0000$<\hat{F}$	b	0,6
			b	0,1	**0,1000**$<\hat{F}$		
			c	0,7	0,7000$>\hat{F}$		
2	0,1000	0,6000	a	0,0	0,1000$<\hat{F}$	c	0,3
			b	0,1	0,1600$<\hat{F}$		
			c	0,7	**0,5200**$<\hat{F}$		
3	0,5200	0,1800	a	0,0	0,5200$<\hat{F}$	b	0,6
			b	0,1	**0,5380**$<\hat{F}$		
			c	0,7	0,6460$>\hat{F}$		
4	0,5380	0,1080	a	0,0	**0,5380**$<\hat{F}$	a	0,1
			b	0,1	0,5488$>\hat{F}$		
5	0,5380	0,0108	a	0,0	0,5380$<\hat{F}$	b	0,6
			b	0,1	**0,5391**$<\hat{F}$		
			c	0,7	0,5456$>\hat{F}$		

```
function y=int_arithm_encoder(x,q);
% x is input data sequence,
% q is cumulative distribution (model)
% y is binary output sequence

% Constants
k=16;
R4=2^(k-2); R2=R4*2; R34=R2+R4;% half,quarter,e
R=2*R2;                          % Precision

% Initialization
Low=0;        % Low
High=R-1;     % High
btf=0;        % Bits to Follow
y=[ ];        % code sequence

% Encoding
for i=1:length(x);
   Range=High-Low+1;
   High=Low+fix(Range*q(x(i)+1)/q(m))-1;
   Low=Low+fix(Range*q(x(i))/q(m));

   % Normalization
   while 1
      if High<R2
         y=[y 0 ones(1,btf)]; btf=0;
         High=High*2+1; Low=Low*2;
      else
         if Low>=R2
            y=[y 1 zeros(1,btf)]; btf=0;
            High=High*2-R+1; Low=Low*2-R;
         else
            if Low>=R4  &  High<R34
               High=2*High-R2+1; Low=2*Low-R2;
               btf=btf+1;
            else
               break;
            end;
         end;
      end;
   end; % while
end; % for

% Completing
if Low<R4
   y=[y 0 ones(1,btf+1)];
else
   y=[y 1 zeros(1,btf+1)];
end;
```

```
function x=int_arithm_decoder(y,q,n);
% y is binary encoded data sequence,
% q is cumulative distribution (model)
% x is output sequence
% n is number of messages to decode

% Constants
k=16; R4=2^(k-2); R2=R4*2; R34=R2+R4; R=2*R2;
m=length(q);

% Start decoding. Reading first k bits
Value=0; y=[y zeros(1,k)];
for ib=1:k
   Value=2*Value+y(ib);
end;

% Initialization
Low=0;  High=R-1;

% Decoding
for j=1:n
   Range=High-Low+1;
   aux=fix(( (Value-Low+1)*q(m)-1)/Range);
   i=1; % message index
   while q(i+1)<=aux, i=i+1; end;
   x(j)=i;
   High=Low+fix(Range*q(i+1)/q(m))-1;
   Low=Low+fix(Range*q(i)/q(m));

   % Normalization
   while 1
      if High<R2
         High=High*2+1; Low=Low*2;
         ib=ib+1;
         Value = 2*Value+y(ib);
      else
         if Low>=R2
            High=High*2-R+1; Low=Low*2-R;
            ib=ib+1;
            Value = 2*Value-R+y(ib);
         else
            if Low>=R4  &  High<R34
               High=2*High-R2+1;  Low=2*Low-R2;
               ib=ib+1;
               Value = 2*Value-R2+y(ib);
            else
               break;
            end;
         end;
      end
   end; % while
end; % for
```