Побуквенное кодирование

Пусть даны алфавит источника $A = \{a_1, a_2, \ldots a_n\}$, кодовый алфавит $B = \{b_1, b_2, \ldots, b_m\}$. Обозначим A^* множество всевозможных последовательностей в алфавите *A*. Множество всех сообщений в алфавите *A* обозначим *S*.

Кодирование F может сопоставлять код всему сообщению из множества ***S*** как единому целому или **строить код сообщения из кодов его частей** (побуквенное кодирование)

.

Пример. $A = \{a_1, a_2, a_3\}$, $B = \{0,1\}$

Побуквенное кодирование $a_1 \to 1001$ $a_2 \to 0$ $a_3 \to 010$

позволяет следующим образом закодировать сообщение

$$a_2 a_1 a_2 a_3 \to 010010010$$

Пример. Азбука Морзе. Входной алфавит – английский. Наиболее часто встречающиеся буквы кодируются более короткими словами:

A$\to$01, B$\to$1000, C$\to$1010, D$\to$100, E$\to$0, F$\to$0010, G$\to$110, H$\to$0000, I$\to$00, J$\to$0111, K$\to$101, L$\to$0100, M$\to$11, N$\to$10, O$\to$111, P$\to$0110, Q$\to$1101, R$\to$010, S$\to$000, T$\to$1, U$\to$001, V$\to$0001, W$\to$011, X$\to$1001, Y$\to$1011, Z$\to$1100.

Побуквенное кодирование задается ***таблицей кодовых слов***: $\sigma = <\alpha_1\to\beta_1,\ \alpha_2\to\beta_2, \dots, \alpha_n\to\beta_n>$, $\alpha_i \in A$, $\beta_i \in B^*$.

Множество кодовых слов букв $\mathbf{V = \{\beta_i\}}$ называется ***множеством элементарных кодов***.

Используя побуквенное кодирование, можно закодировать любое сообщение.

Общий код сообщения складывается из элементарных кодов символов входного алфавита.

Количество букв в слове $a=a_1 \dots a_k$ называется ***длиной слова***.

Пустое слово, не содержащее ни одного символа, **обозначается Λ**.

Если слово $\boldsymbol{a = a_1 a_2}$, то $\boldsymbol{a_1}$ – *начало (**префикс**)* слова a, $\boldsymbol{a_2}$ – *окончание (**постфикс**)* слова a.

Определение. Побуквенный код называется ***разделимым*** (или ***однозначно декодируемым***), если любое сообщение из символов алфавита источника, закодированное этим кодом, может быть однозначно декодировано.

При разделимом кодировании любое кодовое слово единственным образом разлагается на элементарные коды.

Пример. Код $a_1 \to 1001$ $a_2 \to 0$ $a_3 \to 010$

не является разделимым, поскольку кодовое слово **010010** может быть декодируемо двумя способами:

$a_3\, a_3$ или $a_2\, a_1\, a_2$.

Определение. Побуквенный код называется ***префиксным***, если в его множестве кодовых слов ни одно слово не является началом другого, т.е. элементарный код одной буквы не является префиксом элементарного кода другой буквы.

Пример. Код $a_1 \to 1001$ $a_2 \to 0$ $a_3 \to 010$

не является префиксным, поскольку элементарный код буквы a_2 является префиксом элементарного кода буквы a_3.

Утверждение. *Префиксный код является разделимым (однозначно декодируемым).*

Замечание. Разделимый код может быть не префиксным.

Пример. Разделимый, но не префиксный код:

$A = \{ a, b \}$, $B = \{ 0, 1 \}$, $\boldsymbol{a \to 0}$, $\boldsymbol{b \to 01}$

Основные теоремы побуквенного кодирования

Теорема *(Л.Крафт,1949). Для того, чтобы существовал побуквенный двоичный* ***префиксный код с длинами кодовых слов L_1, ..., L_n*** *необходимо и достаточно, чтобы*

$$\sum_{i=1}^{n} 2^{-L_i} \leq 1 \qquad \text{(двоичный случай)}$$

Пример.

Построить префиксный код с длинами ***$L_1 = 1$, $L_2 = 2$, $L_3 = 2$*** для алфавита ***$A = \{ a_1 , a_2 , a_3 \}$***.

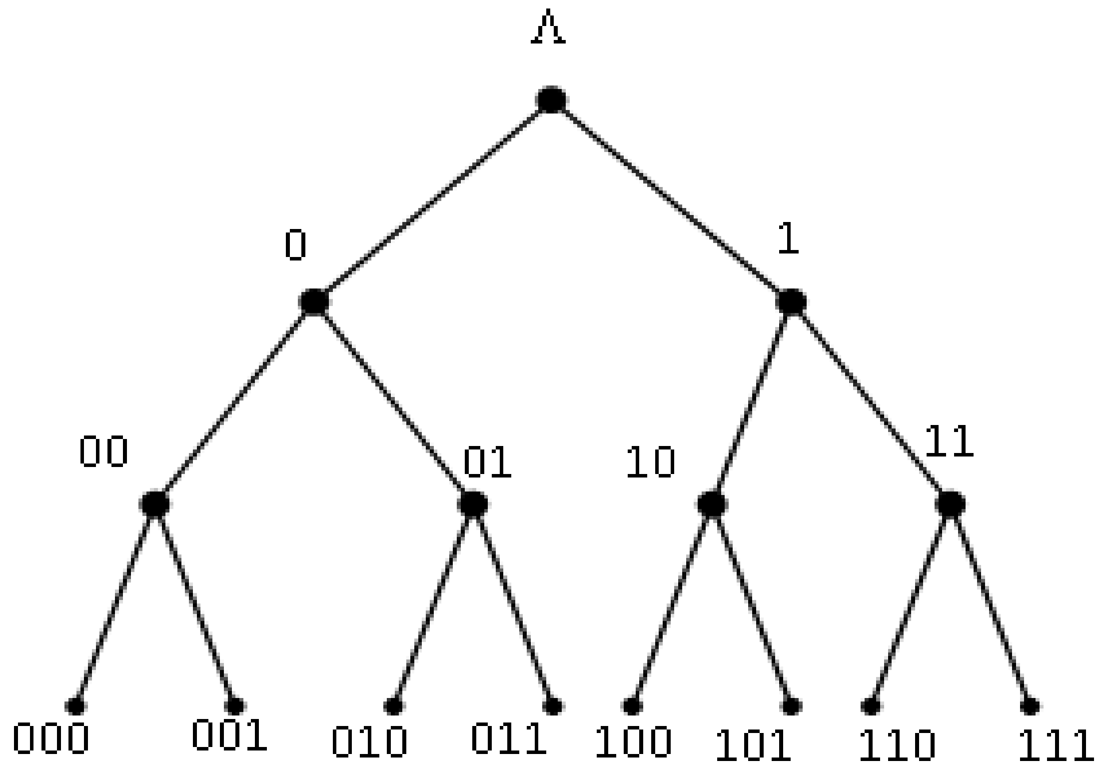
Проверим ***неравенство Крафта*** для заданного набора длин:

$$\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^2} = 1$$

Неравенство Крафта выполняется и, следовательно, **префиксный код** с таким набором длин кодовых слов **существует**.

Неравенство было выведено Леоном Крафтом в своей магистерской дипломной работе в 1949 году.

Рассмотрим полное двоичное дерево. Каждая вершина закодирована последовательностью нулей и единиц.



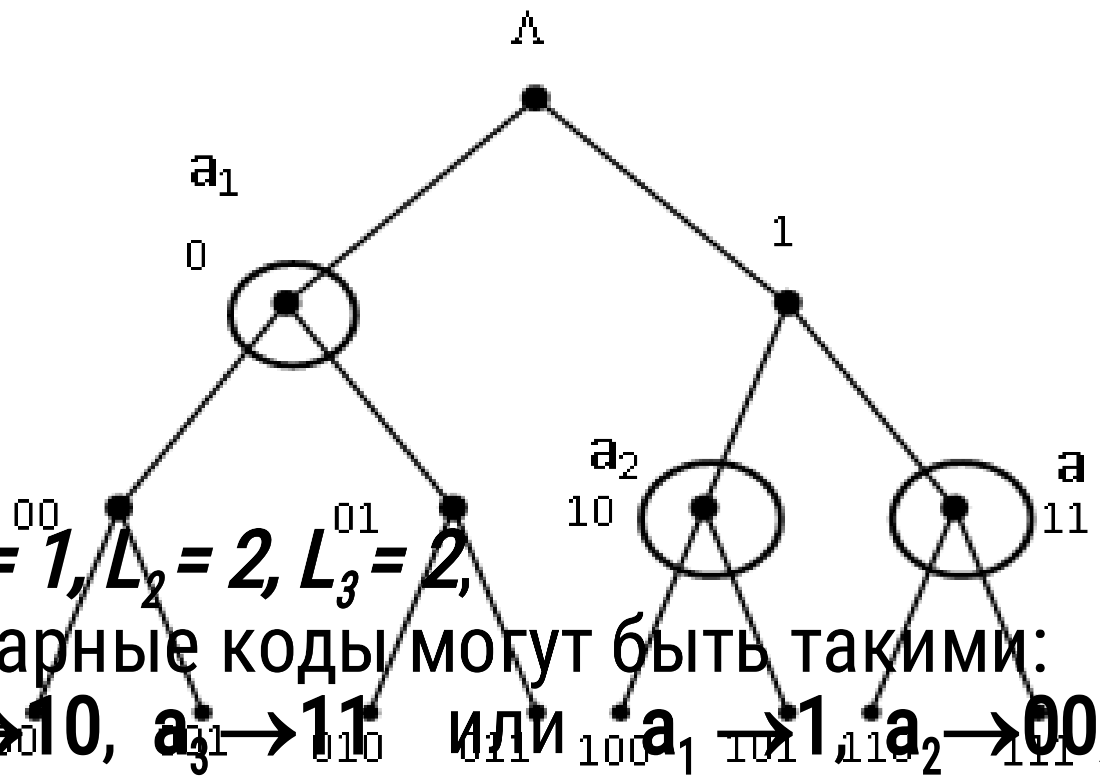
Пусть длины кодовых слов упорядочены по возрастанию

$$L_1 \leq L_2 \leq \ldots \leq L_n.$$

Выберем в двоичном дереве вершину V_1 на уровне L_1.

Уберем поддерево с корнем в вершине V_1.

В оставшемся дереве возьмем вершину V_2 на уровне L_2 и удалим поддерево с корнем в этой вершине и т.д. Последовательности, соответствующие вершинам $V_1, V_2, \ldots, V_n$ образуют ***префиксный код***.



Так как $L_1 = 1, L_2 = 2, L_3 = 2$,

то элементарные коды могут быть такими:

$a_1 \to 0$, $a_2 \to 10$, $a_3 \to 11$ или $a_1 \to 1$, $a_2 \to 00$, $a_3 \to 01$.

Процесс декодирования выглядит следующим образом. Просматриваем полученное сообщение, двигаясь по дереву. Если попадаем в листовую вершину, то выдаем соответствующую букву и возвращаемся в корень дерева и т.д.

Теорема *(Б.МакМиллан,1956). Для того чтобы существовал побуквенный двоичный* ***разделимый код с длинами кодовых слов $L_1, \dots, L_n$****, необходимо и достаточно, чтобы*

$$\sum_{i=1}^{n} 2^{-L_i} \le 1 \qquad \text{(двоичный случай)}$$

Пример. Азбука Морзе – известный побуквенный код:
A→01, B→1000, C→1010, D→100, E→0, F→0010, G→110, H→0000, I→00, J→0111, K→101, L→0100, M→11, N→10, O→111, P→0110, Q→1101, R→010, S→000, T→1, U→001, V→0001, W→011, X→1001, Y→1011, Z→1100.

Неравенство МакМиллана для азбуки Морзе не выполнено:

$$2 \cdot \frac{1}{2^1} + 4 \cdot \frac{1}{2^2} + 8 \cdot \frac{1}{2^3} + 12 \cdot \frac{1}{2^4} = 3\frac{3}{4} > 1$$

Следовательно, **этот код не является разделимым**.

???

На самом деле
в азбуке Морзе имеются **дополнительные элементы – паузы** между буквами (и словами), которые **позволяют декодировать сообщение**.

Эти дополнительные элементы **определены неформально**, поэтому прием и передача сообщений (особенно с высокой скоростью) является некоторым искусством, а не простой технической процедурой.

Пусть имеется **дискретный вероятностный источник** , порождающий символы алфавита $\boldsymbol{A = \{ a_1, a_2, \dots , a_n\}}$ с вероятностями $\boldsymbol{p_1, p_2, \dots, p_n}$.

Основной характеристикой источника

является ***энтропия***, которая представляет собой

среднее значение количества информации

в сообщении источника и определяется выражением (для двоичного случая):

$$H(p_1, \dots, p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i .$$

Энтропия характеризует **меру неопределенности выбора** для данного источника.

Пример 1. Пусть $A = \{a_1, a_2\}$, $p_1 = 0$, $p_2 = 1$.

Источник может породить **только символ a_2**, **неопределенности нет**, энтропия $H(p_1, p_2) = 0$.

Пример 2. Пусть $A = \{a_1, a_2\}$, $p_1 = 1/2$, $p_2 = 1/2$.

Источник **с равновероятными символами** имеет **максимальную** энтропию $H(p_1, p_2) = 1$.

Для практических применений важно, чтобы **коды сообщений** имели по возможности **наименьшую длину**.

Основной характеристикой неравномерного кода является **количество символов**, затрачиваемых **на кодирование одного сообщения**.

Определение. Пусть имеется разделимый побуквенный код для источника, порождающего символы алфавита **A = {$a_1, \dots, a_n$}** с вероятностями $p_i = P(a_i)$, состоящий из n кодовых слов с длинами $L_1, \dots, L_n$ в алфавите **{0,1}**.

Средней длиной кодового слова называется величина

$$L_{cp} = \sum_{i=1}^{n} p_i L_i ,$$

которая показывает **среднее число кодовых букв на одну букву источника**.

Определение. ***Избыточностью кода*** называется **разность** между средней длиной кодового слова и энтропией источника сообщений

$$r = L_{cp} - H(p_1, \ldots, p_n).$$

Избыточность является показателем **качества кода**, оптимальный код обладает минимальной избыточностью.

Задача эффективного неискажающего сжатия заключается в построении кодов с наименьшей избыточностью, у которых средняя длина кодового слова **близка к энтропии** источника.

К таким кодам относятся **классические коды *Хаффмана*, *Шеннона*, Фано, Гилберта-Мура и арифметический код**.

Взаимосвязь между **средней длиной** кодового слова и **энтропией** дискретного вероятностного источника при побуквенном кодировании выражает следующая теорема:

Теорема (К.*Шеннон*). *Для источника с алфавитом* $A = \{a_1, \ldots, a_n\}$ *и вероятностями* $p_i = P(a_i)$, *и любого разделимого побуквенного кода средняя длина кодового слова* ***всегда*** *не меньше энтропии*

$$L_{cp} \geq H(p_1, \ldots, p_n)$$

и ***можно построить разделимый побуквенный код****, у которого средняя длина кодового слова превосходит энтропию не больше, чем на единицу:*

$$L_{cp} < H(p_1, \ldots, p_n) + 1$$

Для доказательства построим **код Шеннона** с длинами кодовых слов

$$L_i < -\log p_i + 1.$$

Построение кода Шеннона:

1) Упорядочим символы исходного алфавита **A = {*a*$_1$, *a*$_2$, …, *a*$_n$}** по убыванию их вероятностей:

$$p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n.$$

2) Вычислим величины Q_i, которые называются ***кумулятивными вероятностями***:

$$Q_0 = 0,\ Q_1 = p_1,\ Q_2 = p_1 + p_2,\ Q_3 = p_1 + p_2 + p_3,\ \ldots,\ Q_n = 1.$$

3) Представим Q_i **в двоичной системе счисления** и возьмем в качестве кодового слова первые $\lceil -\log p_i \rceil$ знаков после запятой.

Для вероятностей, представленных **в виде десятичных дробей**, удобно определить **длину кодового слова L_i** из соотношения:

$$\frac{1}{2^{L_i}} \le p_i < \frac{1}{2^{L_i - 1}}, \; i = 1, \ldots, n.$$

Пример. Пусть дан алфавит **A = {a_1, a_2, a_3, a_4, a_5, a_6}** с вероятностями **p_1= 0.36, p_2= 0.18, p_3= 0.18, p_4= 0.12, p_5= 0.09, p_6= 0.07**.

Проверим, удовлетворяют ли L_i **неравенству Крафта**?

$$\frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^4} + \frac{1}{2^4} = \frac{11}{16} < 1$$

Неравенство выполняется и, следовательно, **префиксный код** с таким набором длин кодовых слов **существует**.

Пример кода Шеннона

a_i	P_i	Q_i	L_i	**кодовое слово**
a_1	$1/2^2 \le \mathbf{0.36} < 1/2$	0	2	00
a_2	$1/2^3 \le \mathbf{0.18} < 1/2^2$	0.36	3	010
a_3	$1/2^3 \le \mathbf{0.18} < 1/2^2$	0.54	3	100
a_4	$1/2^4 \le \mathbf{0.12} < 1/2^3$	0.72	4	1011
a_5	$1/2^4 \le \mathbf{0.09} < 1/2^3$	0.84	4	1101
a_6	$1/2^4 \le \mathbf{0.07} < 1/2^3$	0.93	4	1110

Вычислим **среднюю длину кодового слова**:

$L_{cp} = 0.36 \cdot 2 + (0.18 + 0.18) \cdot 3 + (0.12 + 0.09 + 0.07) \cdot 4 = \mathbf{2.92}$

и сравним ее с энтропией источника сообщений.

Энтропия источника:

$$H(p_1,\ldots,p_6) = -\,0.36\cdot\log 0.36 - 2\cdot 0.18\cdot\log 0.18 -$$

$$-\,0.12\cdot\log 0.12 - 0.09\cdot\log 0.09 - 0.07\log 0.07 = \mathbf{2.37}$$

Для построенного кода:

$$L_{cp} < H(p_1,\ldots,p_6) + 1$$

$$\mathbf{2.92 < 2.37 + 1},$$

что полностью соответствует утверждению теоремы Шеннона.

Условные обозначения в алгоритме:

n – количество символов исходного алфавита

P – массив вероятностей, упорядоченных по убыванию

Q – массив для величин Q_i

L – массив длин кодовых слов

C – матрица элементарных кодов

Алгоритм построения кода Шеннона

```
P[0] := 0, Q[0] := 0
DO ( i := 1, …, n)
      Q[i] := Q[i-1] + P[i]
       L[i] := -⌈log₂ P[i] ⌉
OD
DO ( i := 1, …, n)
      DO ( j := 1, …, L[i] )
   Q[i-1] := Q[i-1] · 2
   C[i, j] :=⌊Q[i-1]⌋
   IF (Q[i-1] >1)  Q[i-1] := Q[i-1] -1  FI
      OD
OD
```

__Пример__. Пусть имеются **два источника** с одним и тем же алфавитом $A = \{a_1, a_2, a_3\}$ и разными вероятностными распределениями $P_1 = \{1/3, 1/3, 1/3\}$, $P_2 = \{1/4, 1/4, 1/2\}$, которые кодируются одним и тем же кодом:

$$\sigma = \left\langle a_1 \to 10,\ a_2 \to 000,\ a_3 \to 01 \right\rangle.$$

Средняя длина кодового слова для разных источников будет различной:

$$L_{cp}(P_1) = 1/3 \cdot 2 + 1/3 \cdot 3 + 1/3 \cdot 2 = 7/3 \approx 2.33$$

$$L_{cp}(P_2) = 1/4 \cdot 2 + 1/4 \cdot 3 + 1/2 \cdot 2 = 9/4 = 2.25$$

__Определение__. Побуквенный разделимый код называется ***оптимальным***, если **средняя длина кодового слова** ***минимальна*** среди всех побуквенных разделимых кодов для данного распределения вероятностей символов.

Оптимальный код Хаффмана

Метод был разработан в 1952 г.
Дэвидом Хаффманом (*David A. Huffman; 1925-1999*) – американский профессор в области теории информации.

Оптимальный код Хаффмана обладает минимальной средней длиной кодового слова среди всех побуквенных кодов для источника с алфавитом $A = \{a_1,\dots,a_n\}$ и $p_i = P(a_i)$.

Алгоритм построения оптимального кода Хаффмана

1. Упорядочим символы исходного алфавита $A=\{a_1,\ldots,a_n\}$ по убыванию их вероятностей $\boldsymbol{p_1 \ge p_2 \ge \ldots \ge p_n}$.

2. Если $\boldsymbol{A = \{a_1, a_2\}}$, то $\boldsymbol{a_1 \to 0}$, $\boldsymbol{a_2 \to 1}$.

3. Если $\boldsymbol{A = \{a_1, \ldots, a_j, \ldots, a_n\}}$ и известны коды $\boldsymbol{< a_j \to b_j >}$, $j=1,\ldots,n$, то для алфавита $\boldsymbol{\{a_1, \ldots a_j', a_j'' \ldots, a_n\}}$ с новыми символами $\boldsymbol{a_j'}$ и $\boldsymbol{a_j''}$ вместо $\boldsymbol{a_j}$, и вероятностями $\boldsymbol{p(a_j) = p(a_j') + p(a_j'')}$, код символа $\boldsymbol{a_j}$ заменяется на коды $a_j' \to \boldsymbol{b_j 0}$, $\boldsymbol{a_j'' \to b_j 1}$.

Пример. Пусть дан алфавит **A = {a_1, a_2, a_3, a_4, a_5, a_6}**

с вероятностями

p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07.

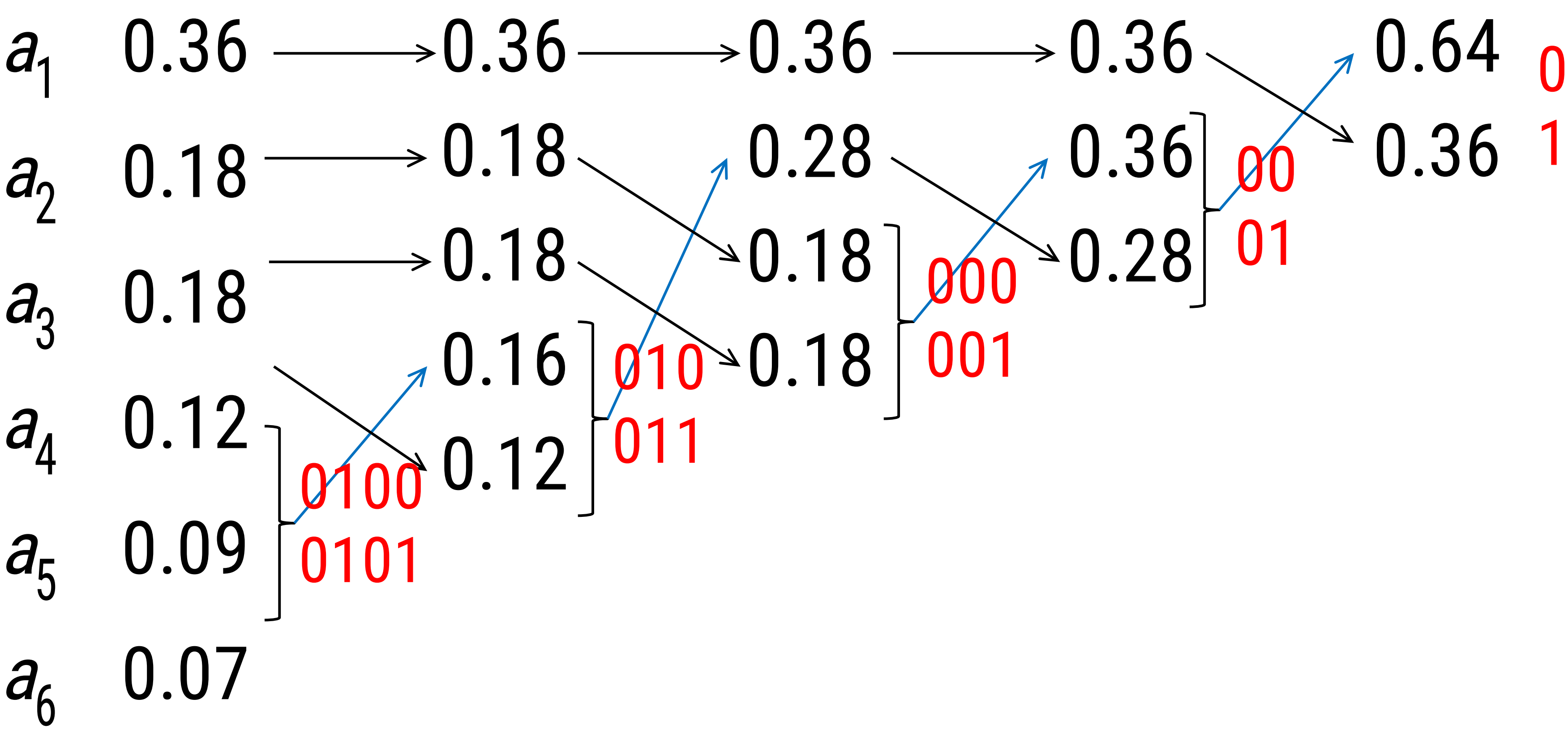
Здесь символы источника уже упорядочены ***по убыванию их вероятностей***.

Будем складывать ***две наименьшие вероятности*** и включать суммарную вероятность на соответствующее место в упорядоченном списке вероятностей до тех пор, пока в списке не останется ***два символа***.

Тогда закодируем эти два символа как **0** и **1**.

Далее кодовые слова достраиваются, как показано на рисунке.

Процесс построения кода Хаффмена



Код Хаффмана

a_i	p_i	L_i	кодовое слово
a_1	0.36	2	1
a_2	0.18	3	000
a_3	0.18	3	001
a_4	0.12	4	011
a_5	0.09	4	0100
a_6	0.07	4	0101

Средняя длина построенного кода Хаффмана:

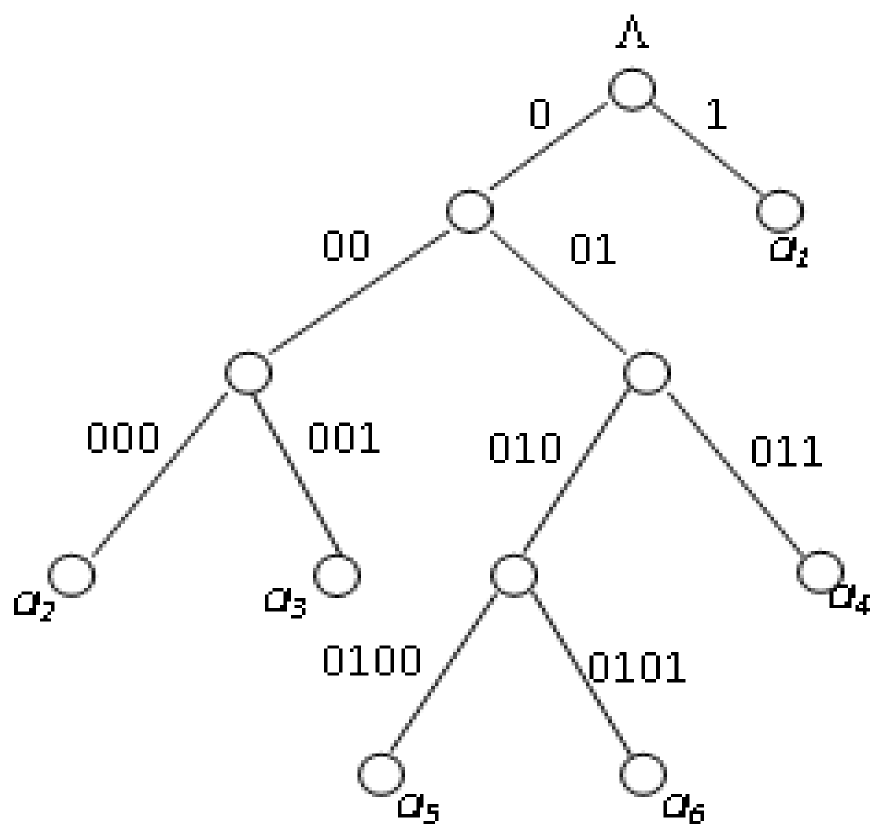
$L_{cp}(P)$=1·0.36 + 3·0.18 + 3·0.18 + 3·0.12 + 4·0.09 + 4·0.07=**2.44**,

Энтропия данного источника:

$H(p_1,\ldots,p_6)$ = − 0.36·log0.36 − 2·0.18·log0.18 −

− 0.12·log0.12 − 0.09·log0.09 − 0.07log0.07 = **2.37**

Код Хаффмана обычно строится и хранится в виде **двоичного дерева**, в листьях которого находятся символы алфавита, а на «ветвях» – 0 или 1. Тогда уникальным кодом символа является **путь от корня дерева к этому символу**, по которому все 0 и 1 собираются в одну уникальную последовательность.



Алгоритм на псевдокоде

Построение оптимального кода Хаффмана (n,P)

n – количество символов исходного алфавита

P – массив вероятностей, упорядоченных по убыванию

C – матрица элементарных кодов

L – массив длин кодовых слов

```
Huffman (n,P)
IF (n=2) C [1,1]:= 0, L [1]:= 1
      C [2,1]:= 1, L [2]:= 1
ELSE  q:= P [n-1]+P [n]
   j:= Up (n,q)                (поиск и вставка суммы)
   Huffman (n-1,P)
   Down (n,j)                  (достраивание кодов)
FI
```

Функция **Up (n,q)** находит в массиве P место, куда вставить число q, и вставляет его, сдвигая вниз остальные элементы.

Алгоритм на псевдокоде

```
Up (n,q)
DO ( i=n-1, n-2,…,2 )
   IF ( P[i-1] ≤ q ) P[i] := P[i-1]
   ELSE j := i
   FI
OD
P [j]:= q
```

Процедура **Down (n,j)** формирует кодовые слова.

Алгоритм на псевдокоде

```
S := C [ j,*]  (запоминание j-той строки матрицы         элем.
кодов в массив S)
L := L[j]
DO ( i := j, …, n-2)
   C[ i,*]:= C[ i+1,*]  (сдвиг вверх строк матрицы C)
   L[i] := L[i+1]
OD
C [n-1,*] := S,     (восстановление префикса
C [n,*] := S        кодовых слов из массива S)
C [n-1,L+1]:=0
C [n,L+1]:=1
L [n-1]:=L+1
L [n]:=L+1
```

Код Фано

(Шеннона-Фано)

Ро́берт Фа́но (*Robert Fano*, 1917—2016) — итальяно-американский учёный в области информатики.

Код Фано является префиксным почти оптимальным кодом, для которого $L_{cp} < H(p_1, \ldots, p_n) + 1$,

Построение кода Фано

Упорядоченный по убыванию вероятностей список букв алфавита источника делится на две части так, чтобы суммы вероятностей букв, входящих в эти части, как можно меньше отличались друг от друга.

Буквам первой части приписывается 0, а буквам из второй части – 1. Далее также поступают с каждой из полученных частей.

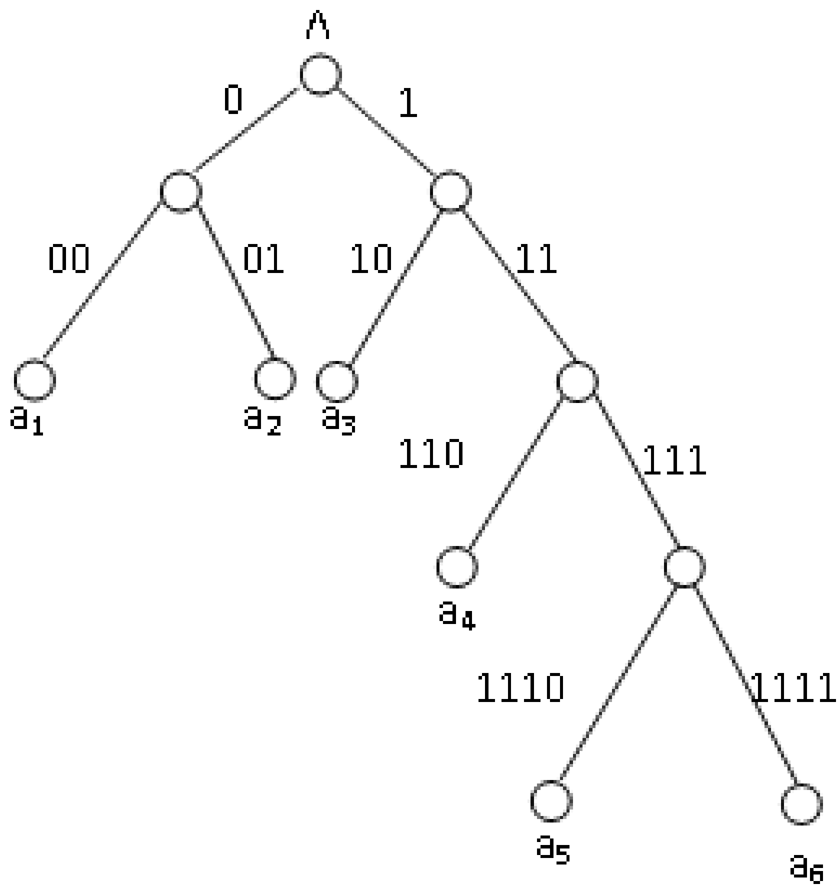
Процесс продолжается до тех пор, пока весь список не разобьется на части, содержащие по одной букве.

Пример. Пусть дан алфавит A = {a_1, a_2, a_3, a_4, a_5, a_6} с вероятностями p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07.

Код Фано

a_i	P_i	кодовое слово				L_i
a_1	0.36	**0**	**0**			2
a_2	0.18	**0**	**1**			2
a_3	0.18	**1**	**0**			2
a_4	0.12	**1**	**1**	**0**		3
a_5	0.09	**1**	**1**	**1**	**0**	3
a_6	0.07	**1**	**1**	**1**	**1**	4

Кодовое дерево для кода Фано



Полученный код является **префиксным** и **почти оптимальным** со средней длиной кодового слова

$\boldsymbol{L_{cp}} = 0.36 \cdot 2 + 0.18 \cdot 2 + 0.18 \cdot 2 + 0.12 \cdot 3 + 0.09 \cdot 4 + 0.07 \cdot 4 = \mathbf{2.44}$

Энтропия источника: $\boldsymbol{H(p_1,\ldots,p_6) = 2.37}$, $\mathbf{2.44 < 2.37 + 1}$

Алгоритм на псевдокоде

Построение кода Фано

Условные обозначения:

P – массив вероятностей символов алфавита
L – левая граница обрабатываемой части массива P
R – правая граница обрабатываемой части массива P
k – длина уже построенной части элементарных кодов
C – матрица элементарных кодов
Length – массив длин элементарных кодов
S_L – сумма элементов первой части массива
S_R – сумма элементов второй части массива.

Алгоритм на псевдокоде

Построение кода Фано

```
Fano (L,R,k)
IF ( L<R )
   k := k+1
   m := Med (L,R)
   DO ( i := L, …, R )
      IF ( i ≤m ) C[i,k]:=0,  Length[i]:= Length[i]+1
      ELSE         C[i,k]:=1,  Length[i]:= Length[i]+1
      FI
   OD
   Fano (L,m,k)
   Fano (m+1,R,k)
FI
```

Функция **Med** находит медиану части массива P, т.е. такой индекс ***L ≤ m ≤ R***, что минимальна величина

$$\left| \sum_{i=L}^{m} p_i - \sum_{i=m+1}^{R} p_i \right|$$

Med (L,R)

S_L:= 0

DO (i:= L, …, R-1)

   S_L:=S_L+ P[i]

OD

S_R:= P[R]

m:= R

DO ($S_L \geq S_R$)

   m:=m-1

   S_L:= S_L - P [m]

   S_R:= S_R+ P [m]

OD

Алфавитный код Гилберта – Мура

Е. Н. Гилбертом и Э. Ф. Муром был предложен метод построения алфавитного кода, для которого

$$L_{cp} < H(p_1, \ldots, p_n) + 2 .$$

Пусть символы алфавита некоторым образом упорядочены, например, $\mathbf{a_1 \le a_2 \le \ldots \le a_n}$.

<u>**Определение**</u>. Код называется ***алфавитным***, если кодовые слова лексикографически упорядочены, т.е.

$$\sigma(a_1) \le \sigma(a_2) \le \ldots \le \sigma(a_n) .$$

Построение кода Гилберта-Мура

1. Вычислим величины Q_i, i=1,n:

$$Q_1 = p_1/2,$$

$$Q_2 = p_1+p_2/2,$$

$$Q_3 = p_1+p_2+p_3/2,$$

…

$$Q_n = p_1+p_2+\ldots+p_{n-1}+p_n/2.$$

2. Представим суммы Q_i в двоичном виде. $\lceil -\log p_i \rceil + 1$

3. В качестве кодовых слов возьмем

   младших бит в двоичном представлении Q_i, .

Пример. Пусть дан алфавит A = $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07.

Код Гилберта-Мура

a_i	P_i	Q_i	L_i	кодовое слово
a_1	$1/2^3 \le 0.18$	0.09	4	0001
a_2	$1/2^3 \le 0.18 < 1/2^2$	0.27	4	0100
a_3	$1/2^2 \le 0.36 < 1/2^1$	0.54	3	100
a_4	$1/2^4 \le 0.07$	0.755	5	11000
a_5	$1/2^4 \le 0.09$	0.835	5	11010
a_6	$1/2^4 \le 0.12$	0.94	5	11110

Средняя длина кодового слова:

$\boldsymbol{L_{cp}}$= 4·0.18+4·0.18+3·0.36+5·0.07+5·0.09+5·0.12=**3.92<2.37+2**

Арифметический код

Идея арифметического кодирования была впервые предложена ***П. Элиасом*** (*P. Elias*).

В арифметическом коде кодируемое сообщение разбивается **на блоки постоянной длины**, которые затем кодируются отдельно.

<u>При увеличении длины блока</u> средняя длина кодового слова стремится к энтропии, **однако**

возрастает сложность реализации алгоритма и уменьшается скорость кодирования и декодирования.

Арифметическое кодирование позволяет получить **произвольно малую избыточность** при кодировании достаточно больших блоков входного сообщения.

Идея арифметического кода

Пусть дан источник, порождающий буквы из алфавита $\boldsymbol{A}=\{a_1,a_2,\dots,a_n\}$ с вероятностями $\boldsymbol{p_i} = P(a_i)$.

Необходимо закодировать некую последовательность символов данного источника $\boldsymbol{S} = x_1 x_2 x_3 x_4 \dots$

1. Вычислим кумулятивные вероятности $\boldsymbol{Q_0}, \boldsymbol{Q_1}, \dots, \boldsymbol{Q_n}$:

$Q_0=0$

$Q_1=p_1$

$Q_2=p_1+p_2$

$Q_3=p_1+p_2+p_3$

…

$Q_n=p_1+p_2+\dots+p_n=1$

Идея арифметического кода

2. Разобьем интервал $[Q_0, Q_n)$ - интервал $[0,1)$ - так, чтобы каждой букве исходного алфавита соответствовал свой интервал, равный ее вероятности:

$a_1 \quad [Q_0, Q_1)$

$a_2 \quad [Q_1, Q_2)$

$a_3 \quad [Q_2, Q_3)$

$a_4 \quad [Q_3, Q_4)$

...

$a_n \quad [Q_{n-1}, Q_n)$

3. В процессе кодирования будем выбирать интервал, соответствующий текущей букве исходного сообщения, и снова разбивать его пропорционально вероятностям исходных букв алфавита.

Идея арифметического кода

На примере кодирования последовательности $a_3a_2a_3$

Идея арифметического кода

Постепенно происходит ***сужение интервала*** до тех пор, пока не будет закодирован последний символ кодируемого сообщения.

Двоичное представление любой точки, расположенной внутри интервала, и будет **кодом** исходного сообщения.

Для однозначного декодирования
исходной последовательности достаточно взять
$\lceil \log(r_k) \rceil$ разрядов
двоичной записи любой точки из этого интервала,

где r_k – длина интервала после кодирования k символов источника.

Обозначения:

l_i – нижняя граница отрезка, соответствующего i-той букве исходного сообщения;

h_i – верхняя граница этого отрезка;

r_i – длина отрезка $[l_i, h_i)$, т.е. $r_i = h_i - l_i$

Начальные значения:

$$l_0 = Q_0 = 0, \quad h_0 = Q_k = 1, \quad r_0 = h_0 - l_0 = 1$$

Границы интервала для кодируемой буквы:

$$l_i = l_{i-1} + r_{i-1} \cdot Q_{m-1}$$

$$h_i = l_{i-1} + r_{i-1} \cdot Q_m$$

где m – порядковый номер кодируемой буквы в алфавите источника, $m = 1,..,n$.

Окончательная длина интервала равна произведению вероятностей всех встретившихся символов.

Начало интервала зависит от порядка расположения символов в кодируемой последовательности.

Пример. Кодирование бесконечной последовательности $X = a_3 a_2 a_3 a_1 a_4 \ldots$ в алфавите $A = \{a_1, a_2, a_3, a_4\}$ с вероятностями $p_1 = 0.1$, $p_2 = 0.4$, $p_3 = 0.2$, $p_4 = 0.3$.

Вычислим кумулятивные вероятности Q_i:

$$Q_0 = 0,$$

$$Q_1 = p_1 = 0.1,$$

$$Q_2 = p_1 + p_2 = 0.5,$$

$$Q_3 = p_1 + p_2 + p_3 = 0.7,$$

$$Q_4 = p_1 + p_2 + p_3 + p_4 = 1$$

Границы интервала для первого символа кодируемого сообщения $\boldsymbol{a_3}$:

$$l_1 = l_0 + r_0 \cdot Q_2 = 0 + 1 \cdot 0.5 = 0.5,$$

$$h_1 = l_0 + r_0 \cdot Q_3 = 0 + 1 \cdot 0.7 = 0.7,$$

Длина интервала после символа $\boldsymbol{a_3}$:

$$r_1 = h_1 - l_1 = 0.7 - 0.5 = 0.2.$$

Границы интервала для второго символа кодируемого сообщения $\boldsymbol{a_2}$:

$$l_2 = l_1 + r_1 \cdot Q_1 = 0.5 + 0.2 \cdot 0.1 = 0.52,$$

$$h_2 = l_1 + r_1 \cdot Q_2 = 0.5 + 0.2 \cdot 0.5 = 0.6,$$

Длина интервала после символа $\boldsymbol{a_2}$:

$$r_2 = h_2 - l_2 = 0.6 - 0.52 = 0.08$$

Последовательность интервалов
для сообщения $a_3a_2a_3a_1a_4$:

В начале $[0.0, 1.0)$
После просмотра a_3 $[0.5, 0.7)$
После просмотра a_2 $[0.52, 0.6)$
После просмотра a_3 $[0.56, 0.576)$
После просмотра a_1 $[0.56, 0.5616)$
После просмотра a_4 $[0.56112, 0.5616)$

Кодом последовательности $a_3a_2a_3a_1a_4$ будет двоичная запись любой точки из интервала $[0.56112, 0.5616)$, например, 0.56112.

Для однозначного декодирования потребуется $\lceil \log_2(r_5) \rceil = \lceil \log_2(0.00048) \rceil = 12$ двоичных разрядов,

Выводы

- При арифметическом кодировании сообщение представляется вещественными числами в интервале [0, 1).
- По мере кодирования сообщения отображающий его интервал уменьшается, а количество битов для представления интервала возрастает.
- Очередные символы сообщения сокращают величину интервала в зависимости от значений их вероятностей.
- Более вероятные символы делают это в меньшей степени, чем менее вероятные, и следовательно, добавляют меньше битов к результату.

Алгоритм на псевдокоде

Арифметическое кодирование

m – порядковый номер кодируемой буквы в алфавите источника

l_0:=0; h_0:=1; r_0:=1; i:=0

DO (not EOF)

C:=Read( ) (читаем следующий символ из файла)

i:=i+1

  DO (j=1,…,n)

   IF (C= a_j) m:=j FI

  OD

$l_i = l_{i-1} + r_{i-1} \cdot Q[m-1]$

$h_i = l_{i-1} + r_{i-1} \cdot Q[m]$

$r_i = h_i - l_i$

OD

Арифметическое декодирование

В начале декодирования известен конечный интервал, например, ***[0.56112; 0.5616)*** или любое число из этого интервала, например, ***0.56112***.

Сразу можно определить, что первым закодированным символом был a_3, т. к. число ***0.56112*** лежит в интервале ***[0.5; 0.7)***, выделенном символу a_3.

Затем в качестве интервала берется ***[0.5; 0.7)*** и в нем определяется диапазон, соответствующий числу 0***.56112***.

Это интервал ***[0.52, 0.6)***, выделенный символу a_2 и т.д.

Для декодирования **необходимо знать**

- количество закодированных символов и
- исходные вероятности символов.

Алгоритм на псевдокоде

Арифметическое декодирование

length – количество закодированных символов,
value – значение из входного файла.

l_0:=0; h_0:=1; r_0:=1;

value:=ReadCode(); (читаем код из файла)

DO (i=1,…,length)

DO (j=1,…,n)

$l_i = l_{i-1} + r_{i-1} \cdot Q\,[j-1]$

$h_i = l_{i-1} + r_{i-1} \cdot Q\,[j]$

$r_i = h_i - l_i$

IF ((l_i <= value) и (value< h_i)) OD FI

OD

Write (a[i]) (пишем символ в выходной файл)

OD

При реализации арифметического кодирования возникают **две проблемы**:

- необходима арифметика с плавающей точкой теоретически неограниченной точности;
- результат кодирования становится известен только после окончания входного потока.

Для решения этих проблем

реальные алгоритмы работают с целыми числами и <u>оперируют с дробями, числитель и знаменатель</u> которых являются целыми числами

(например,

знаменатель равен 10000h = 65536, l_0=0, h_0=65535).

При этом

- С ***потерей точности*** можно бороться, отслеживая сближение l_i и h_i и умножая числитель и знаменатель представляющей их дроби на одно и то же число (например на 2).
- С ***переполнением сверху*** можно бороться, записывая старшие биты l_i и h_i в файл только тогда, когда они перестают меняться (т.е. уже не участвуют в дальнейшем уточнении интервала).